Rozprawy
Monografie 350

350 ROZPRAWY
MONOGRAFIE

MAREK PLUTA

Synteza dźwięku w reprodukcji i wykonawstwie muzyki

WYDAWNICTWA AGH

KRAKÓW 2019

DISSERTATIONS
MONOGRAPHS 350

MAREK PLUTA

Sound Synthesis for Music Reproduction and Performance

AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY PRESS KRAKOW 2019

Published by AGH University of Science and Technology Press

Author of the monograph is an employee of
AGH University of Science and Technology
Faculty of Mechanical Engineering and Robotics
Department of Mechanics and Vibroacoustics
al. A. Mickiewicza 30
30-059 Krakow, Poland

Desktop publishing: *Marek Pluta*
Technical editor: *Magdalena Grzech*

ISBN 978-83-66016-69-9
ISSN 0867-6631

Wydawnictwa AGH (AGH University of Science and Technology Press)
al. A. Mickiewicza 30, 30-059 Kraków
tel. 12 617 32 28, 12 636 40 38
e-mail: redakcja@wydawnictwoagh.pl
www.wydawnictwa.agh.edu.pl

Contents

MAREK PLUTA
Sound Synthesis for Music Reproduction and Performance

Summary

Sound synthesis has a history dating from the turn of XIX-th and XX-th century. At present, synthesizers are commonly utilised in music, and they surpass traditional instruments in the abilities related to the control over parameters of generated sound. This quality allows to imitate and substitute existing instruments, as well as leads to artistic experiments in the area of new means of expression. At the same time, synthesis techniques remain firmly based on signal processing, and even more so on mechanics. After all, not only methods of synthetic sound generation, but also characteristics and parameters of the effect achieved, have to be considered and studied in a category of physical, mechanical phenomena.

The monograph attempts to present current state of knowledge regarding sound synthesis methods in two main areas of their musical applications: reproduction of music from a symbolic score, and live music performance. The survey of synthesis methods is based on the author's proposal for their classification, which takes account of the fundamental nature of the sound production principle. Apart from traditional, often studied methods, the monograph presents a number of new or less known methods, hitherto rarely discussed in books, such as concatenative or neural audio synthesis. The objective of the survey was to present the subject in a manner helpful and suitable for a sound engineer that either attempts to use, or to design sound synthesizers.

On the basis of this broad survey, the monograph presents two new, author's methods of sound synthesis. The first one, phrase assembling synthesis, is an attempt aimed towards realistic reproduction of musical scores. It combines selected features of sampling and concatenative synthesis with a score interpreting sequencer supplemented with performance rules simulation algorithms. The second method is based on a certain paradox, i.e. a numerical simulation of infeasible instruments. The key idea is to design a mathematical model of a sound producing object that purposefully breaks selected rules or exceeds feasible ranges of parameters. Instruments created in this manner retain some features of real objects, which facilitates their intuitive control, while the sound they produce manifests new, often complex and interesting properties, which have been presented and analysed in the monograph.

MAREK PLUTA

Synteza dźwięku w reprodukcji i wykonawstwie muzyki

Streszczenie

Historia współczesnej syntezy dźwięku sięga przełomu XIX i XX wieku. W tym czasie syntezatory znalazły szerokie zastosowania w muzyce, przewyższając tradycyjne instrumenty w zakresie możliwości kontroli parametrów wytwarzanego dźwięku. Ta cecha pozwoliła im z jednej strony imitować i zastępować istniejące instrumenty, a z drugiej umożliwiła artystyczne eksperymenty w sferze poszukiwań nowych środków wyrazu. Jednocześnie same techniki syntezy pozostają bardzo silnie zakorzenione w takich dziedzinach wiedzy jak przetwarzanie sygnałów, ale przede wszystkim mechanika. To ostatecznie w kategorii fizycznego, mechanicznego zjawiska muszą być rozpatrywane zarówno same metody syntetycznego wytwarzania dźwięku, jak również cechy i parametry osiągniętego efektu.

Monografia przedstawia aktualny stan wiedzy dotyczącej metod syntezy dźwięku wykorzystywanych w dwóch głównych obszarach jej muzycznych zastosowań: do reprodukcji muzyki na podstawie zapisu symbolicznego oraz do wykonywania muzyki na żywo. Przegląd metod syntezy oparto na autorskiej propozycji ich podziału, uwzględniającej istotę zasady wykorzystanej do generowania dźwięku. Poza metodami tradycyjnymi, szeroko opisywanymi w literaturze, praca przybliża także wiele nowych lub mniej znanych metod, do tej pory rzadko omawianych w opracowaniach książkowych, takich jak synteza konkatenacyjna czy też metoda oparta na technikach głębokiego uczenia. Autor monografii skonstruował ją tak, by mogła być użyteczna dla inżynierów dźwięku niezależnie od tego, czy zajmują się jedynie wykorzystaniem czy również projektowaniem i budową syntezatorów.

Na tym szeroko zarysowanym tle zaprezentowane są dwie nowe, autorskie metody syntezy. Pierwsza z nich, metoda montażu frazy, reprezentuje grupę metod przeznaczonych przede wszystkim do realistycznej reprodukcji zapisu nutowego. Łączy ona wybrane cechy metody konkatenacyjnej i samplingowej z elementami sekwencera wspartego algorytmami symulacji technik wykonawczych i interpretacyjnych. Druga z metod opiera się na pewnym paradoksie, czyli numerycznej symulacji instrumentów nierealizowalnych. Polega ona na modelowaniu fizycznym obiektów wytwarzających dźwięk, w których celowo łamie się określone zasady bądź też przekracza możliwe do zaistnienia zakresy parametrów. Powstałe w efekcie instrumenty zachowują pewne cechy obiektów rzeczywistych, co ułatwia ich intuicyjną kontrolę. Jednocześnie jednak wytwarzany przez nie dźwięk ma nowe, często złożone i interesujące właściwości, które zostały zaprezentowane i poddane analizie.

1. Introduction

Since ancient times music has been an essential and influential component of human culture, that apart from artistic and utilitarian aspect, always triggered human ingenuity to find, and later design, objects producing sounds with appealing features. What exactly is musically appealing, depends on the purpose of music, its form, and state of its evolution, with rhythmic, dynamic, melodic, harmonic, sonoristic, or other features considered primary. Thus a progress in instrument design and production methods does not slow down, even though at times, certain instruments reach such level of refinement and perfection that further improvements do not seem possible – the violin could be the very example.

Despite maturing, crystallising designs, science and technology constitute a source of ideas that gives momentum to try entirely new approaches. From the very beginning instruments were based on principles of mechanics, often quite sophisticated, so that even today full understanding of the underlying phenomena regarding sound production in certain instruments may be debatable to some extent, hampering accurate modelling attempts and prediction of instruments behaviour. More recently, with the earliest endeavours dating to XVIII-th century and devices such as *Denis d'or* and *clavecin électrique*, musical instruments started to employ principles of electricity. Initially, electricity served as an aid to otherwise mechanical designs, but in time it has become the principal source of signal in instruments such as *the theremin*, *the ondes martenot*, or *the Trautonium*. Yet before such instruments were able to gain acceptance comparable to their traditional counterparts, they have spawned a new generation of musical instruments – the sound synthesizers.

1.1. Definition of Sound Synthesis

The area of sound synthesis is located at the intersection of science, technology, and arts. The science is the primary source of methods, the technology is responsible for implementations, and the arts give it a purpose. Each of them is focused on a different aspect, so when it comes to defining the essence of sound synthesis the result varies depending on the point of view.

The most fundamental statement is that the synthesis is a process of producing sound [485]. Such description, however, is clearly too broad. In musical applications, a sound synthesizer is considered to be an electronics-based device or a computer program that applies an algorithm substantiating a certain principle to produce sound. Well defined process of sound generation allows to accurately predict its outcome, giving synthesizers an advantage over traditional and electronic instruments when it comes to reproducing existing, or designing new sounds. New sounds, in turn, may be the cause of a common understanding of synthesizers as 'artificial', contrary to 'natural' instruments, based on principles of mechanics.

The term 'artificial' is one of three adjuncts that often complement definitions of sound synthesis, with the remaining two being 'electronic' and 'algorithmic' [521, 470]. However, none of these is actually immanent for sound synthesis, and does not provide an explanation on the concept – it is quite easy to give prominent counterexamples.

Even though majority of older, standalone synthesizers can be viewed as electronic musical instruments, some of the oldest, such as *the Telharmonium*, are electromechanical. One can even consider a pipe organ as a variant of entirely mechanical additive synthesizer. Other exceptions are quite new and of a different kind: contemporary synthesizers are based on pure algorithms implemented in a software form, while electronics serves as a mere framework.

An algorithm seems to be a crucial part of a synthesizer, yet again, *the Telharmonium* proves otherwise. There is no actual algorithm in its electro-mechanical design, yet it obviously applies the additive synthesis method, implemented as a mechanism. Therefore, a sound producing algorithm is not an universal and distinguishing feature of sound synthesis.

The term 'artificial' is the most questionable of the three. In general understanding it refers to being humanly contrived [364]. However this would make virtually all musical sounds artificial, since they would have been produced on purpose. A rough approach would be to put a division line between sounds generated mechanically and electronically. In early years of electronic music that would be sufficient and quite precise. But since that time many electronic instruments have become as much acknowledged as their mechanical counterparts, and their sound is considered no longer artificial. Moreover, mechanical instruments are recorded and reproduced with electronic devices, which does not attach the 'artificial' quality to their sound. Thus understanding of the term constantly evolves, and it might be more arguably attributed to new, previously unknown sounds. However, many synthesizers are part of a musical tradition now, and their sound is anything but novel. Therefore, it is not artificial.

With neither of the aforementioned terms being able to define sound synthesis, a distinctive feature has to be looked for elsewhere. With vast variety of synthesized sounds, signal features alone cannot serve the purpose. However, if synthesizers are considered the third generation of musical instruments, one can attempt to isolate – by comparison – the essence of their operation. Mechanical instruments are centred around a physical object and its properties, with the actual physical principles governing production of sound being of secondary concern – a result of the design. Electronic instruments focus on properties of the signal. Here the sound production principle is more important, yet still it is the design of instrument circuits that is of

primary concern. Again, the principle is the result. Sound synthesizers interchange these priorities: the design is a consequence of a chosen sound production principle. Thus the synthesizers are centred around a precisely defined principle.

All things considered, the **sound synthesis** can be defined as **a process leading to production of sound, that is centred around a clearly defined principle**. The medium and method of implementing the principle is irrelevant. There are no assumptions regarding properties of produced sounds. Even though, the definition allows to distinguish between synthesizers and other instruments, and does not exclude non-electronic devices. Following a definition of sound synthesis, one can define the sound synthesizer as any device, either electronic, mechanical, or purely information-based, that performs the sound synthesis using one or more synthesis methods.

1.2. Taxonomy of Synthesis Methods

Discussing detailed differences among numerous synthesis methods and their implementations requires establishing some sort of systematics. Approaches to the problem vary from hierarchical to a group-oriented organisations with diverse and sometimes inconsistent criteria. Beneath the inconsistencies lies the question, whether to consider synthesis on a practical or theoretical basis. A practical approach would start with features of particular synthesizers and infer conclusions from their similarities. A theoretical one would consider general principles of synthesis methods.

Russ [485] has chosen the practical approach. A hierarchy starts with a fundamental division into analogue and digital methods. Thus the main differentiation is based on the implementation. There are three types of analogue methods: subtractive, additive, and wavetable. On a digital side a number of types is larger, including: frequency modulation, wavetable, sample replay, additive, samples and synthesis (S&S), physical modelling, and software synthesis. The apparent flaw of such classification is placement of some methods on both branches. A general division into analogue and digital may better suit sound synthesizers, while in case of methods alone it disregards their actual principles. It has to be considered though, that often a single synthesizer employs several methods. Finally, software synthesis seems misplaced. While it is digitally based, it is also general enough to be classified on the same level as analogue and digital branches, since it can implement any method, such as additive or sampling.

A different approach has been employed by Roads [470], who presents somewhat different list of methods that includes: additive, subtractive, wavetable, sampling, physical modelling, wave terrain, granular, modulation, formant, waveform segment, graphic, and stochastic synthesis. However, since the book concerns computer music, the list is narrowed down to digital synthesis methods only. Moreover there is no move towards grouping methods that display similar properties nor an attempt to establish any apparent hierarchy.

Schwarz [502] classifies synthesis methods using a hierarchy tree presented in Figure 1.1. It emphasises a role of concatenative synthesis, and locates it on the highest branch of hierarchy. While there may be a merit to such classification regarding gen-

erality and inclusiveness of synthesis principles, it does not reflect importance and actual impact of the concatenative method over the methods it supposedly includes.

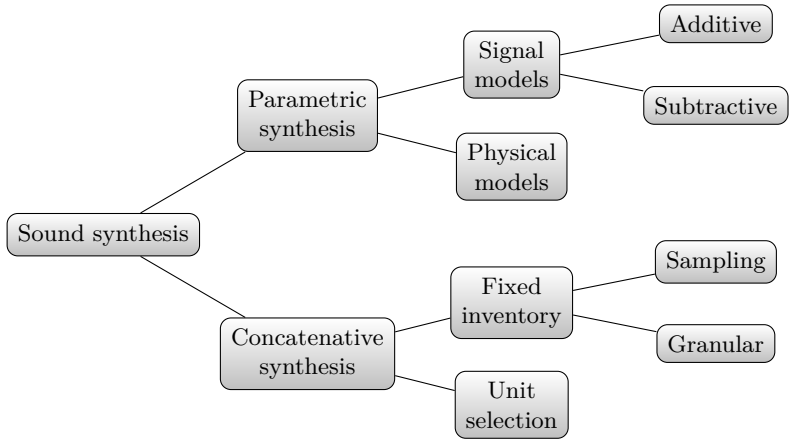


Figure 1.1. Classification of sound synthesis methods, according to Schwarz [502]

A more balanced taxonomy has been proposed by Smith [521], even though, similarly to Roads, he considers digital techniques only. He provides a long, detailed list of methods, and groups them into four classes dealing with: processed recordings, spectral models, physical models, and abstract algorithms. Furthermore, he predicts migration of recording based methods towards spectral group, and eventual disappearance of abstract methods due to lack of analysis support, and resulting difficulty in obtaining sounds that would be musically appealing. Thus with time only two categories shall remain: physical modelling and spectral modelling. He refers to the former, based on the mathematical description of existing instruments, as model of the source, an to the latter, based on the perception of sound, as model of the receiver.

The author of this monograph proposes to expand the original four-group hierarchy proposed by Smith with an additional, highest level, and divide all synthesis methods into direct or indirect, as presented in Figure 1.2). Principles applied in direct methods aim primarily at producing sound, whereas principles of indirect methods concentrate on various models and ideas that may produce sound. Methods belonging to the first group allow to control sound parameters and properties directly. In the second group control affects an intermediate layer. It may lead to more intuitive control, like in physical modelling synthesis, where a musician understands properties of the instrument. However, it may also cause the opposite effect, with an example of modulation methods, where a single parameter can impact several sound characteristics in a non obvious manner.

Direct methods include spectral and waveform based techniques. Spectral techniques operate primarily in frequency domain, and their principles are close to the human perception of sound. As such, they are inherently intuitive. Waveform-based techniques operate directly on sound recordings, thus they usually are very efficient.

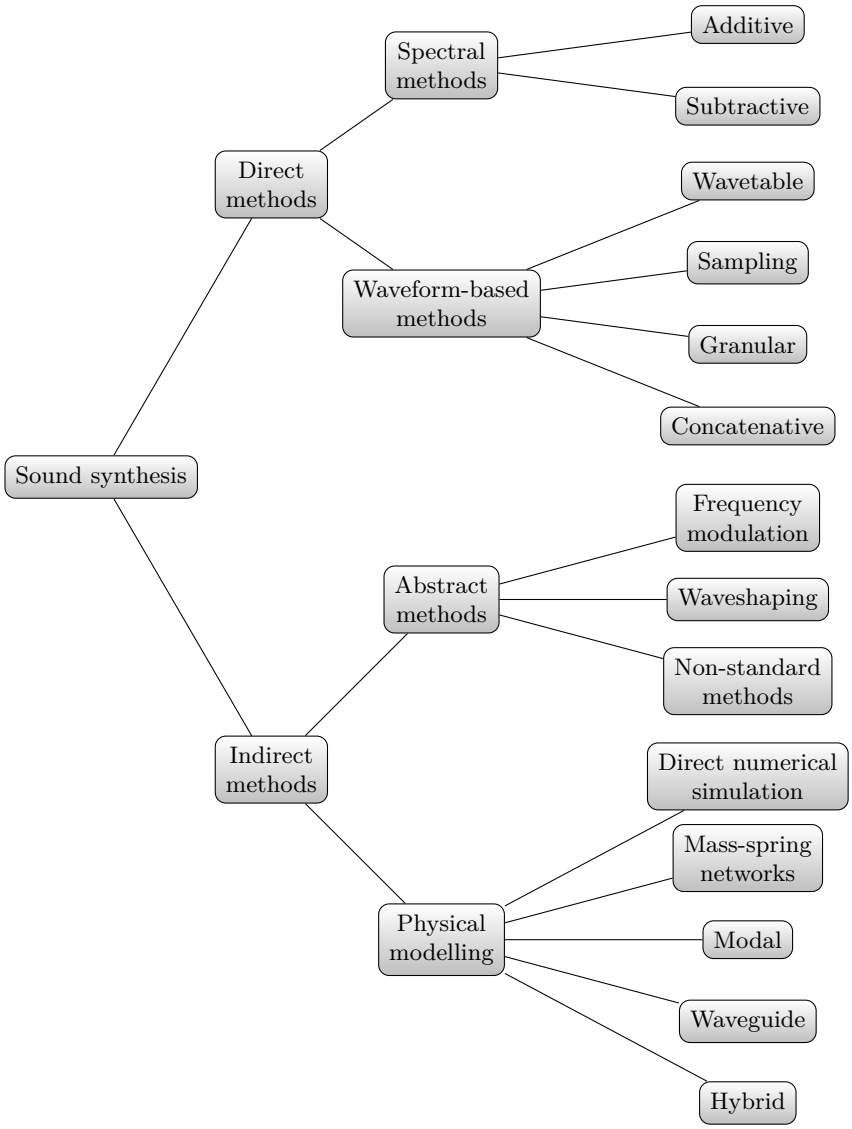


Figure 1.2. Proposed taxonomy of sound synthesis methods

Indirect methods consist of abstract and physical modelling techniques. The first group aggregates approaches based on various principles that are neither inherently associated with sound signal, nor with musical instruments. With no associations and their rules imposed, abstract methods are particularly well suited for sound experiments [179] and may often produce new, intriguing sounds. The second group

applies various numerical modelling approaches to simulate musical instruments – mostly acoustic, but recently analogue electronic as well [399].

1.3. The Purpose and Scope of the Monograph

Sound synthesis has a history dating from the turn of XIX-th and XX-th century. At present, synthesizers are commonly utilised in music. They surpass traditional instruments in the abilities related to the sound control. They can be more precise, repeatable if needed, and unlike traditional instruments that affect timbre through articulation or dynamics, synthesizers allow to adjust it in a more fundamental, unrestrained manner. Thus timbre becomes another controllable quality, providing a composer with an additional, fully featured dimension besides duration, loudness, and pitch. With a deep control over produced sound, a synthesizer can imitate, and sometimes substitute another instrument or a group of instruments. Synthesis is not limited to imitation though, but is a powerful tool allowing to experiment with entirely new sounds.

This monograph has two main objectives. **The first objective is to present current state of sound synthesis methods** in a manner useful for a sound engineer that either attempts to use, or design sound synthesizers. Established works [521, 470, 557, 485] are becoming outdated. They do not include new methods such as concatenative synthesis or neural audio synthesis. These methods have been recently made available due to progress in computer and information technology. Moreover, previously described synthesis methods have progressed as well – notably additive and physical modelling methods, that can make use of additional processing power facilitated by multi core processors and principally by graphics processing units, recently turned into general purpose processing devices. Therefore, the author attempts to update and complete this knowledge.

The second objective is to present the author's contribution into the filed of sound synthesis, in the form of two new synthesis methods: phrase assembling synthesis, and infeasible instruments. Phrase assembling synthesis is an attempt at combining selected features of sampling and concatenative synthesis, aimed towards realistic reproduction of musical scores. It may be considered as a system consisting of a synthesizer and a score interpreting sequencer. It is based on several areas of knowledge, including musical acoustics, sound engineering, orchestration, musical analysis, music performance, and information technology. Infeasible instruments aim at the opposite end of sound synthesis applications, i.e. at live performance and design of new sounds for musical purposes. The method is based on the principles of physical modelling synthesis, and relies on simulation of musical instruments that retain some properties of the original objects, but are modified in various ways that would prevent them from being built in a physical form. Thus they combine two important features: they can be controlled by a performer in an intuitive manner, but at the same time they produce sounds with features uncommon for any existing instrument. Due to high complexity of underlying computations, presented

implementation of infeasible instruments makes use of graphics processing units to perform finite difference simulations.

The monograph includes five chapters and a summary. The first chapter defines the sound synthesis and introduces author's view on the taxonomy of sound synthesis methods. The second chapter presents sound synthesis techniques belonging to a group of direct methods. This includes spectral methods and waveform based methods that are studied with regards to their abilities at reproducing particular musical features. A large part of the chapter is devoted to concatenative synthesis, that has not been widely described in the literature, but its implementations are gaining attention, and it shares some mechanisms with the author's method of phrase assembling. The third chapter presents indirect methods of sound synthesis, including abstract an physical modelling techniques. A section devoted to abstract methods presents a few interesting techniques that have not been widely described in literature, including a new method based on deep learning. Part devoted to physical modelling methods is dominated by finite difference approximations, as the most universal of presented physical approaches, and at the same time as a computational basis for the infeasible instruments method. The fourth chapter presents the author's method of phrase assembling synthesis, from the concept, through design and realisation, to preliminary tests of a prototype implementation. The fifth chapter presents the author's concept and implementation of the infeasible instruments method. It includes a section devoted to programming sound synthesizers based on finite difference schemes, using graphics processing units to simulate large models in real time, thus allowing to produce a performance synthesizer. The following sections present implementations and properties of several infeasible instruments. Finally, the conclusions of the monograph outline the author's accomplishments and summarise the place of both new methods with regards to current state of sound synthesis.

2. Direct Methods

2.1. Spectral Methods

2.1.1. Modular View on Elements of a Synthesizer

Operation of sound synthesizers may be analysed using a convenient abstraction layer, introduced to group parameters related to certain tasks, and to clarify function of particular synthesizer elements. Blocks performing simple tasks, such as signal generation or a specific type of signal modification, are referred to as **unit generators** (UG). Symbols representing UGs are commonly used in synthesizer diagrams.

The concept of UGs can be traced down to large modular analogue synthesizers, where clear flow of control data was vital for handling of a device. UGs however, are not limited to modular appliances – they can describe arrangement of elements and data flow in closed-architecture synthesizers as well. It may be possible to use the concept of UGs referring to methods other than spectral, although it is the additive and particularly the subtractive synthesis, that may be considered as based entirely on such model.

In analogue synthesizers the data is represented by control voltage (CV) and gate signals. Due to analogue origins nomenclature of UGs is based on this form of data, even though digital synthesizers use different systems and protocols, such as MIDI. Direct usage of UGs naming scheme in digital domain is therefore not precise, since it is not a voltage that is the control signal. Nevertheless, the concept became so popular, that analogue terms were brought to digital domain to name digital counterparts of analogue UGs. Traces of this nomenclature are still present in contemporary music programming languages.

The list of the most basic UGs consists of [485]:

- VCO – voltage controlled oscillator,
- VCA – voltage controlled amplifier,
- VCF – voltage controlled filter,
- LFO – low frequency oscillator,
- EG – envelope generator.

2.1.1.1. Voltage Controlled Oscillator

From the functional point of view VCO is a basic periodic signal generator operating in auditory frequency range. In hybrid and digital synthesizers it is also referred to as DCO (digitally controlled oscillator) to emphasize a fact, that control data is digital. A more general OSC (oscillator) term is also used. A main input of a VCO is frequency. There can be additional inputs for amplitude, and for frequency modulation. If the latter is controlled by LFO it produces a *vibrato* effect. VCOs may have input for connecting output from other VCOs, e.g. for a synchronisation purpose. Finally, when applied in subtractive synthesis, there is a waveform selector, with optional input allowing to adjust its shape (e.g. pulse width).

VCOs applied in additive synthesizers generate sine output. Such signal may not be useful for subtractive synthesis due to lack of harmonics, but it may serve different purposes. For instance, it can be mixed with other waveshapes, or can modulate other signals. Apart from sine signal, analogue VCOs typically generate triangle, square, sawtooth (Fig. 2.1), or pulse waveshapes (Fig. 2.2), and their variants [485]. In hybrid or digital implementations this set may be much larger.

User of a subtractive synthesizer needs to know spectra of these signals in order to be able to determine a possible effect of their filtering. The practical ability to distinguish their characteristic auditory features, such as spectral envelope or absence of even harmonics, can be trained using various timbre solfege tools, as described by Pluta and Kleczkowski [444]. Auditory skill should be followed by precise knowledge of respective spectra.

A Fourier series is a convenient source of required information, including amplitudes and phases of partials, in easy to implement form. In fact, for triangle, square and sawtooth signals phase information can be greatly simplified. The only phase shifts required are inversions (shifts by π), and they can be substituted by negative amplitudes. Therefore all signals considered can be represented by weighted sums of sine components.

The simplest case is a **sawtooth** signal (Fig. 2.1, bottom plot). It can be produced using the following formula

$$u(t) = \frac{2}{\pi} \sum_{k=1}^{\infty} (-1)^k k^{-1} \sin(2\pi k f_0 t) \tag{2.1}$$

where $u(t)$ is the output value, f_0 is the fundamental frequency, and k is the partial index (harmonic). The output will be limited to $[-1, 1]$ interval. Sawtooth from (2.1) ramps up. If the reverse is required one needs to skip $(-1)^k$ that shifts every second harmonic by π

$$u(t) = \frac{2}{\pi} \sum_{k=1}^{\infty} k^{-1} \sin(2\pi k f_0 t) \tag{2.2}$$

and the result will be a sawtooth that firstly jumps, and then ramps down. In case of discrete signals, a sawtooth, as well as square, triangle, and pulse, should be band-limited. Therefore a sum needs to end on such k that $kf_0 < f_N$ (Nyquist frequency) to avoid aliasing.

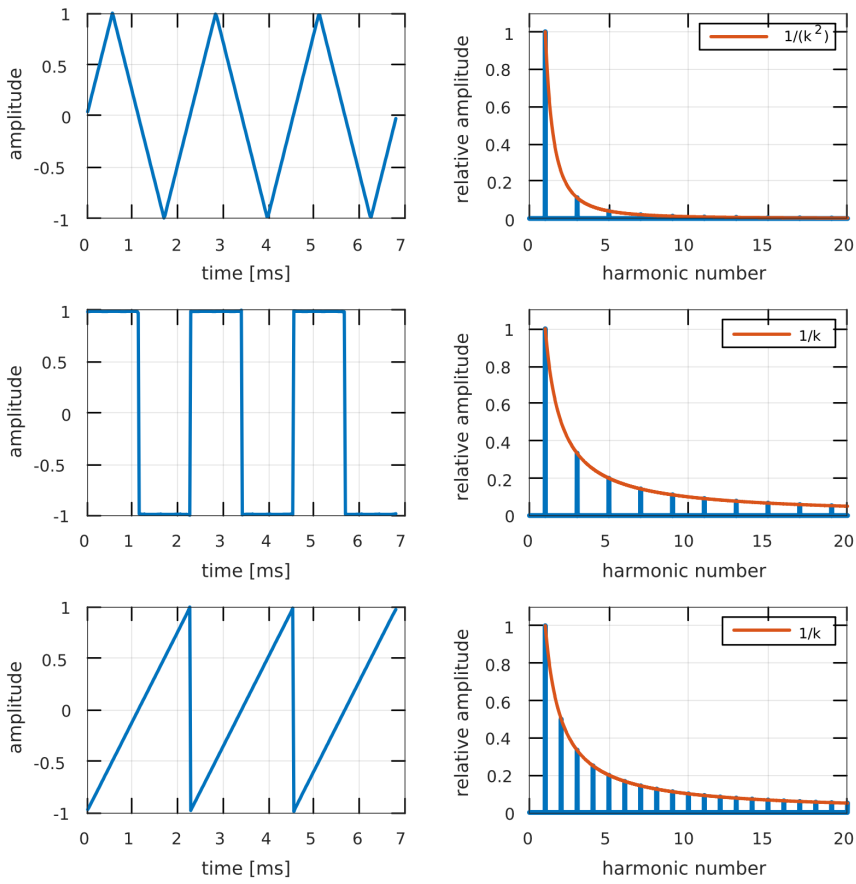


Figure 2.1. Waveshapes (left) and magnitude spectra (right) of a triangle (top plots), square (middle), and sawtooth (bottom) signal with common $f_0 = 440$ Hz; spectra are plotted against harmonic numbers k to emphasize lack of even harmonics in triangle and square signals; red curve on spectra plots represents spectral envelope – common for square and sawtooth signal; harmonics magnitudes are scaled to the magnitude of f_0 partial

A slightly adjusted formula produces a **square** signal that shares a common spectral envelope with a sawtooth, but contains only odd harmonics (Fig. 2.1, middle plot)

$$u(t) = \frac{4}{\pi} \sum_{k=1}^{\infty} (2k-1)^{-1} \sin\left(2\pi \left(2k-1\right) f_0 t\right) \tag{2.3}$$

In the above expression k is still the partial index, but it is no longer index of a harmonic, due to even harmonics missing.

Spectral envelope of a **triangle** signal makes its partials decay faster than previous two. Similarly to square signal, a triangle also has only odd harmonics (Fig. 2.1, top plot), and can be expressed by the following formula

$$u(t) = \frac{8}{\pi^2} \sum_{k=1}^{\infty} (-1)^{k-1} (2k-1)^{-2} \sin\left(2\pi \left(2k-1\right) f_0 t\right) \tag{2.4}$$

Every second partial of a triangle signal is phase-shifted by π, hence the coefficient $(-1)^{k-1}$ is introduced. If this is changed to $(-1)^k$, instead of ascending from $t = 0$, a triangle will first descend – its waveform will be amplitude-reversed.

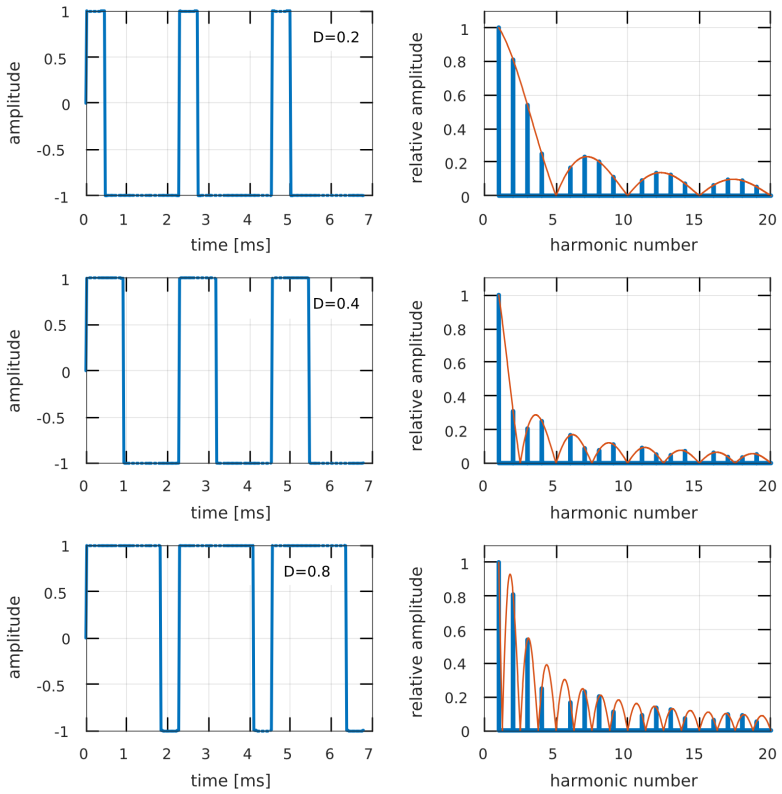


Figure 2.2. Waveshapes (left) and magnitude spectra (right) of a pulse signal ($f_0 = 440$ Hz) with three values of duty cycle D; spectra are plotted against harmonic numbers; red curve represents spectral envelope, according to (2.7); magnitudes in spectra are scaled to the magnitude of f_0 partial; even though envelopes for $D = 0.2$ and $D = 0.8$ are different, discrete spectra are identical, and it is the same for all pairs: D and $(1 - D)$

An interesting case is a **pulse** signal, also referred to as a pulse train (Fig. 2.2). In subtractive synthesizers pulses are rectangular. Unlike sine, triangle, square, and sawtooth, all of which have fixed spectra, spectrum of a pulse signal can be adjusted

through manipulation of a pulse width W_p, which has a dimension of time and is expressed in seconds. W_p can be substituted by a duty cycle $D \in (0, 1)$ that controls the ratio of a pulse width to a signal period T

$$D = \frac{W_p}{T} \tag{2.5}$$

For $D = 0.5$ pulse train becomes a square signal, as in (2.3). The following expression presents the Fourier series expansion of a pulse signal with adjustable duty cycle

$$u(t) = (2D - 1) + \sum_{k=1}^{\infty} \frac{4}{k\pi} \sin(\pi k D) \cos(2\pi k f_0 t - \pi k D) \tag{2.6}$$

In the formula above pulse starts at $t = 0$. If this is not required initial phase $\pi k D$ under cosine can be skipped, and point $t = 0$ will divide the pulse in half – the waveshape will have a y-axis symmetry. Amplitudes of partials change according to

$$A_k = \frac{4}{k\pi} \sin(\pi k D) \tag{2.7}$$

and D^{-1} determines position of the first zero in a spectral envelope expressed as a harmonic number. If it is not integer, all partials are present, although attenuated according to the envelope. Otherwise, partials where $kD \in \mathbb{Z}$ are missing from spectrum. For decreasing D the first zero in envelope moves towards higher harmonics (Fig. 2.3). Therefore very narrow pulses produce almost flat spectrum.

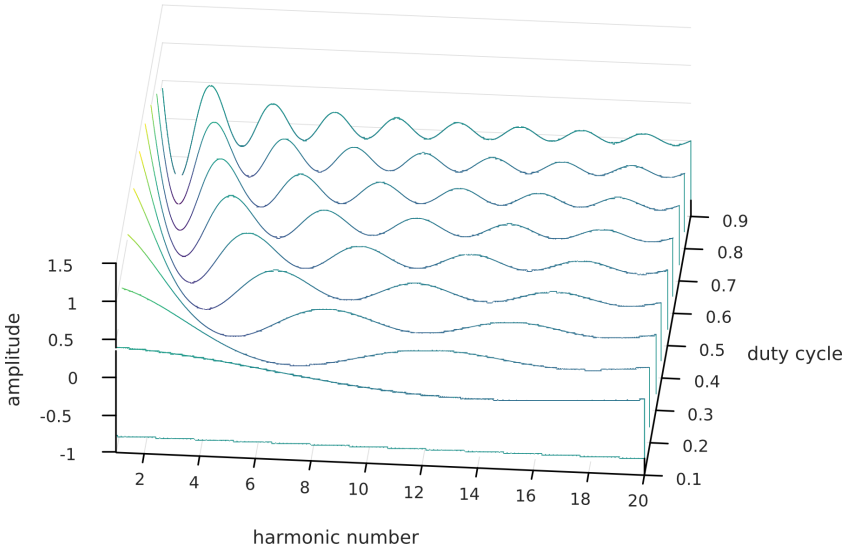


Figure 2.3. Waterfall plot displaying amplitude envelopes of pulse signal partials for duty cycle values between 0.1 and 0.9; negative amplitude values represent phase shifts by π

Signals produced by both, analogue and digital oscillators, only approximately match the ideal waveshapes, and vary depending on particular implementation. In digital implementations periodic signal can be synthesized directly from Fourier series, using additive synthesis. It prevents aliasing, but is computationally inefficient, and may pose a problem in devices with low computing power. Periodic signals can also be produced by transforming sawtooth signal, generated using fast and simple modulo counter [487]. Such method introduces aliasing, but it can be efficiently attenuated using Bandlimited Impulse Train (BLIT) [536] or Bandlimited Step Functions (BLEP) [79] that modify a signal in time domain by adding ripple characteristic to the effect of bandlimited additive synthesis. Curiously, contemporary digital implementations of subtractive synthesis often implement BLIT in an attempt to recreate characteristics of analogue oscillators while preventing aliasing in a technique of **virtual analog** [399].

2.1.1.2. Voltage Controlled Amplifier

VCA can change signal gains according to input value. In analogue applications VCAs may have linear or exponential inputs [485]. The former are usually used for amplitude modulation, such as *tremolo* effect. The latter can be connected with output of linear amplitude envelopes. *Tremolo* is produced by controlling VCA through LFO outputting sine or triangle signal. If LFO is replaced by VCO, amplitude modulation becomes fast enough for sidebands to become audible as two additional frequencies for each sine component of the input signal.

2.1.1.3. Voltage Controlled Filter

Due to a different approach to shaping of a signal spectrum, basic variant of additive synthesis does not utilise VCFs. However, filters are the most important modifiers in subtractive synthesis – they directly affect the spectrum, thus shaping a timbre of sound. Various types of filters can be utilised, but for a synthesizer the most important aspect is the frequency response, or attenuation curve. Depending on the response shape VCFs perform low-pass, high-pass, band-pass, or notch filtering – number and type of a block control inputs changes accordingly. Analogue implementations typically utilise two or four pole filter designs, while digital can have more than eight poles [485].

2.1.1.4. Low Frequency Oscillator

Similarly to VCO, LFO is also a periodic signal generator, but with frequencies below the auditory range. It is used to produce effects such as *vibrato*, *tremolo*, or *wah-wah* through modulation of frequency, amplitude or filter parameters. Connection diagrams for these effects are presented in Figure 2.4, while a spectral evolution they cause is shown in Figure 2.5.

In LFO it is the waveshape that is more important than spectrum [485] – the actual waveshape is audible due to its low frequency. Therefore phases of partials play much larger role, and there are e.g. two different sawtooth variants – one in a ramp-up form, and the other in ramp-down (time-inverted). A special type of LFO output is referred to as **sample and hold**. It operates by sampling value of another signal

once in every period of LFO. Sampled value is held until next one arrives. The result is a step-like signal. If a noise is sampled, the pattern is random. In case of a periodic source the result is periodic, with period depending on sampling and sampled signal frequency ratio, though for long periods it can appear as quasi-random.

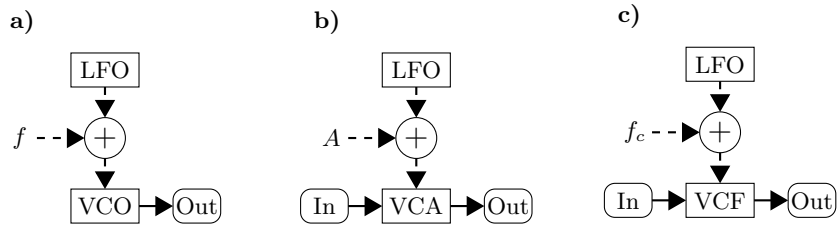


Figure 2.4. Modulation effects attainable through application of LFO: a) *vibrato* – a frequency (f) modulation effect; b) *tremolo* – an amplitude (A) modulation; c) *wah-wah* – a modulation of cut-off frequency (f_c) in low-pass filter

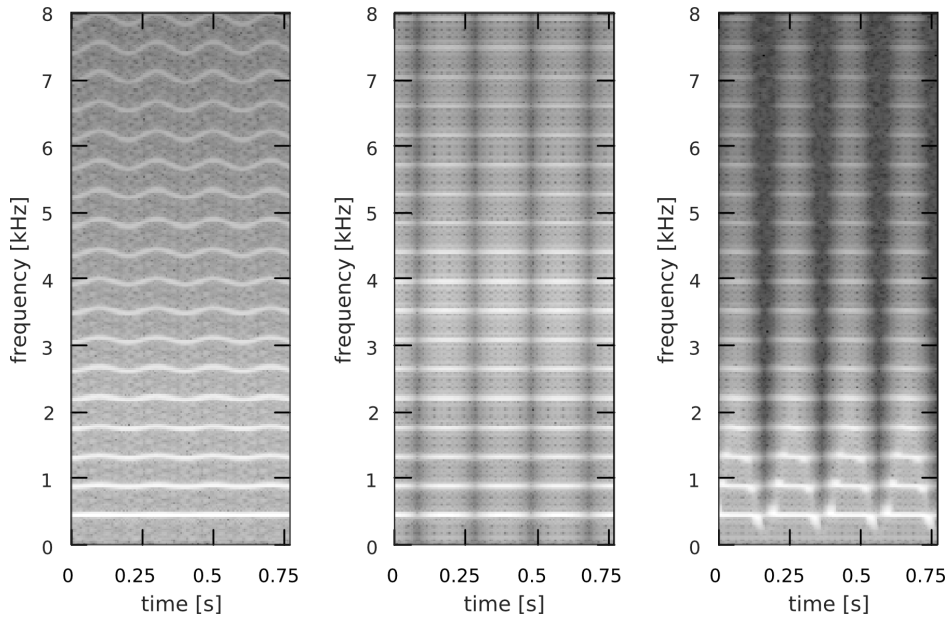


Figure 2.5. Spectral evolution of a periodic signal caused by modulation effects: *vibrato* (left), *tremolo* (middle), and *wah-wah* (right plot); LFO frequency was set to 5 Hz

2.1.1.5. Envelope Generator

An envelope is a curve that outlines extremes of a signal [271]. In sound synthesis, and in spectral methods in particular, the term has a special meaning that may seem reversed: it is not a signal characteristics, but one of control data sources. Initially

envelopes were utilised to introduce time-variability to signal amplitude through control of an amplifier gain. In such role they were very close to the original meaning of a signal envelope. After applying an envelope to control signal amplitude, the actual signal envelope was taking shape of the controlling envelope. When modular approach had been gradually applied to the synthesizers, a link between the envelope and the amplitude has loosened, as the envelope proved to be a convenient and universal source of control data.

Envelope generators produce output that changes over time. Their primary function is similar to LFOs – they are a source of variability applied to selected synthesis parameters related to amplitude, frequency, or spectrum. Though in contrast to LFOs they are not periodic. Envelopes have either fixed duration or their duration can be controlled by an external source – such as a person playing a synthesizer, or a sequencer. Therefore EGs are suitable for controlling evolution of an entire sound event.

An envelope may be given by any time-domain function. It can be arbitrarily provided by a user, sampled from a data source, or defined by some kind of formula. The most common approach is to assemble an envelope out of a small number of ramps or segments. Thus an envelope can be considered a piecewise linear function, or more universally – a piecewise function, since not only linear, but also exponential or logarithmic segments are utilised, which introduces additional parameter related to a segment curvature.

Segments are applied to approximate variability of a selected parameter, in an attempt to reduce amount of control data (Fig. 2.6). This is often the case in the additive synthesis, where envelopes control parameters of signal partials while synthesizing sound of acoustic instruments. In a different approach, applied in the subtractive synthesis and its derivatives, EG produces a universal sequence of segments. A set of standard segment names is presented in Table 2.1. Some envelopes use all of these segments, other ones only a selected subset, and in some cases chosen segments may occur more than once. Sequence of such segments is an attempt to generalise envelope of a musical instrument – with proper values it allows to roughly approximate actual amplitude envelopes of many acoustic instruments. In some EGs not only sustain, but also initial, peak, and final level can be set.

Popular envelope types are presented in Figure 2.7. Their naming scheme seems simple – from the first letters of subsequent segments – but it is not always strictly observed. In a few cases some letters are omitted as obvious. It is important to note that one segment stands out among other parameters listed in Table 2.1. While the majority of parameters control segment duration, sustain controls segment level. Its duration is controlled externally – usually the end of segment is triggered by releasing a controller key. The role of sustain segment is to 'freeze' envelope and keep it at a defined level for as long as required. Therefore if envelope contains such segment, its total duration can be freely adjusted.

The first type – AD envelope (Fig. 2.7) – does not have sustain segment. In such envelopes attack is typically short, and key is released during decay without impact on total envelope duration. However, if a key is released in attack segment, envelope can be shortened. It jumps to decay immediately. Output value though is not set

to peak, but instead starts at value at which attack has been interrupted, and is shortened to keep a decay rate.

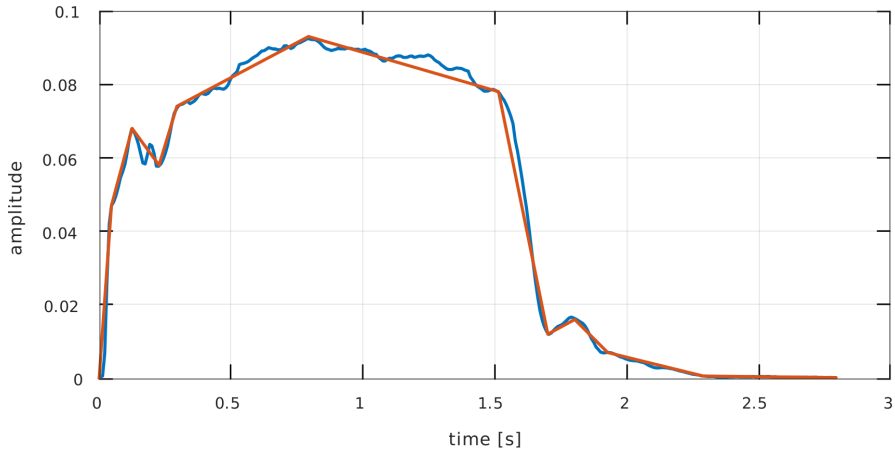


Figure 2.6. Amplitude envelope of a fourth harmonic of a bassoon playing B2 pitch (blue), and its 11-segment approximation (red)

Table 2.1. Common envelope segments

Segment	Parameter	Description
Delay	Time	Before the initial level rise; envelope lag
Attack	Time	Rise from initial to peak level
Hold	Time	Keep at peak level
Decay	Time	Fall to sustain level
Sustain	Level	Keep for an externally controlled amount of time
Release	Time	Fall from sustain to final level

AR envelope (Fig. 2.7b) is the simplest of envelopes that contain sustain segment. Key can be released during sustain, which triggers start of release segment, or during attack. In the latter case AR envelope behaves similarly to AD – attack is interrupted and release starts with current output value. Release time is shortened, while release rate is conserved.

AD envelope can be supplemented with an additional segment. In ADR variant appended release segment allows to change decay slope on the key-off event. In ADS decay is followed by a sustain, and after that – either by a fast release, or by a release with the same time as decay. In contrast to AD, duration of both ADR and ADS can be controlled.

ADSR (Fig. 2.7c) is a more versatile envelope. It is still quite simple, but allows to simulate much broader set of instruments, and therefore is the most popular one. In some implementations it has not one, but two decay segments before the sustain, with

a break point between, where the decay rate changes – it is a rough approximation of exponential or logarithmic curve. If key is released before sustain, the envelope jumps to the final release, just as AD or AR.

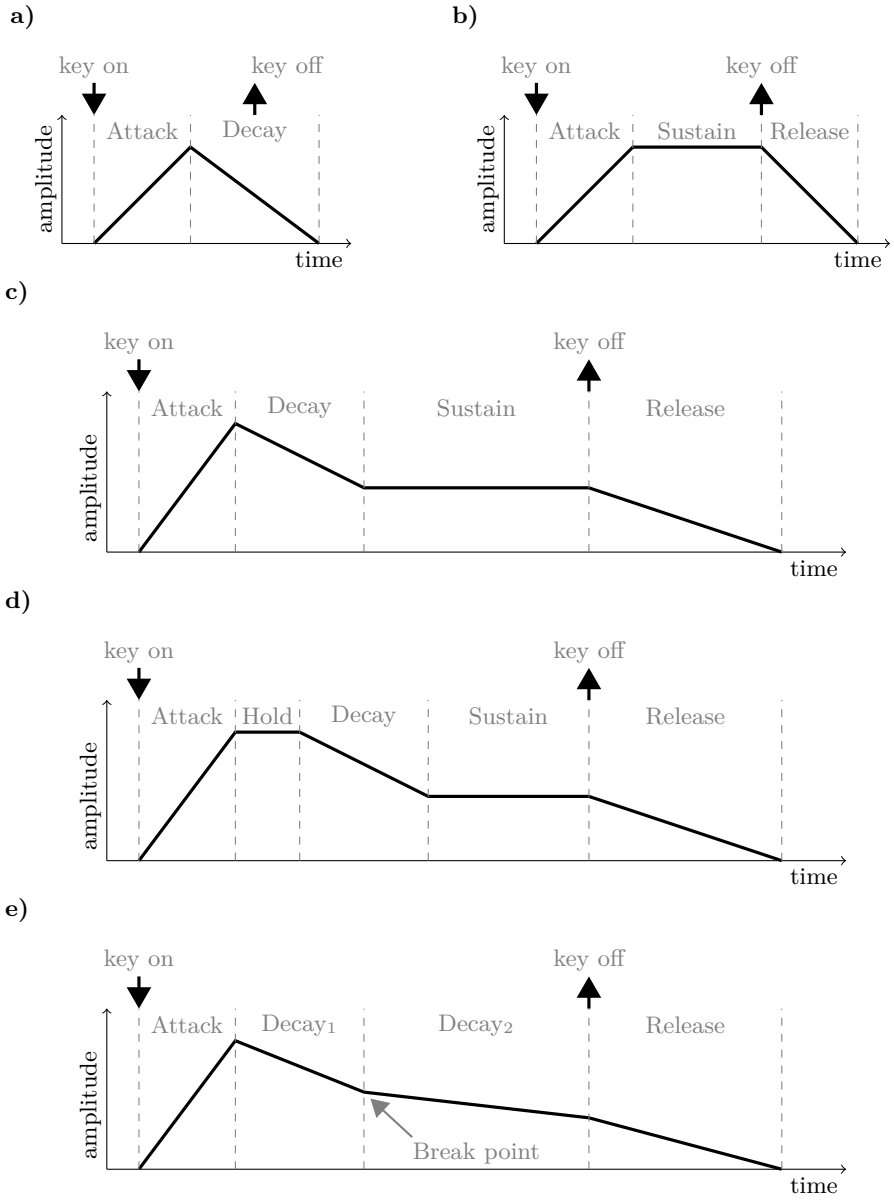


Figure 2.7. Envelope types: a) AD; b) AR; c) ADSR; d) AHDSR; e) ADBDR

Variants of ADSR address its shortcomings in simulating percussive or otherwise decaying envelopes. AHDSR envelope (Fig. 2.7d) is useful when a very rapid attack and decay have to be slightly separated. ADBDR (Fig. 2.7e) is suitable for piano-like sounds [485], where instead of a constant sustain level, a slow decay with externally controlled duration is required. Therefore, similarly to ADR, second decay either does not have predefined duration, only decay rate, or simply has the duration set to a very large value.

There are many more possible segments arrangements. In some implementations envelope can be preceded with a fixed delay. All segments with slope (other than sustain or hold) can have one or more breakpoints. And finally, there are multi-segment envelopes with various sequences of sections.

In monophonic synthesizers, reproducing a single pitch at a time, there is a possibility, that a key is still held in sustain while another one is pressed. In such case EG may restart part of attack starting from a sustain level, and continue through decay to sustain again. Releasing the first key will then be ignored, and only releasing the second will trigger release segment, unless a third is pressed before, and so on. In less common cases EGs can be triggered not by a key, or a *note-on* event, but by LFOs.

Additive synthesizers may use separate envelopes for each signal partial, or for groups of partials. In the most basic case a single envelope can be applied to the sum of all partials. A subtractive synthesizer may have only one EG controlling both, amplifiers and filters. However, it is common that separate EGs control VCA and VCF, and there can be another one modulating VCO frequency.

2.1.2. Additive Synthesis

A musician, familiar with the concept of harmonics, might consider additive synthesis the most intuitive way to synthesize sound of a musical instrument. The method is based on Fourier theory that allows to decompose a periodic or time-bounded signal into the sum or integral of sinusoids. In additive synthesis the analysis process is reversed: sinusoidal components are composed together to produce sound with the desired spectrum [470, 557, 61]. In a typical case sinusoids represent fundamental frequency and harmonic overtones of a pitched sound [233, 202, 438]. Overtones take part in forming the sensation of timbre, yet under some circumstances they can be perceived as separate pitches [299, 385]. What makes the method particularly intuitive is a direct and simple relation between control data and resultant timbre – adjusting parameters of the sound production process leads to predictable auditory results.

The method does not impose any conditions on the component frequencies and their relations. In particular, they need not be in a harmonic ratio, though in pitched sounds it is often assumed that they are. If a single sinusoidal component is described by

$$u(t) = A\cos(2\pi ft + \phi) \tag{2.8}$$

where $u(t)$ is the output value, A_k is the amplitude, f_k is the frequency, and ϕ is the initial phase, than the whole signal takes the following form

$$u(t) = \sum_{k=0}^{M-1} A_k \cos(2\pi f_k t + \phi_k) \tag{2.9}$$

where M is the number of components. Each component can have independent amplitude, frequency, and initial phase. The sum starts from zero to honour a convention, according to which the fundamental frequency is represented by f_0. A schematic diagram of the process is presented in Figure 2.8.

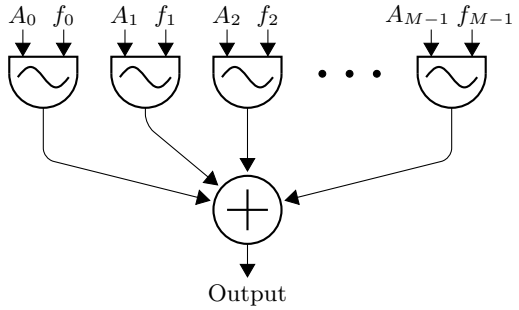


Figure 2.8. A basic variant of additive synthesis with M sinusoidal oscillators in parallel configuration corresponding to formula (2.9); each oscillator has individual frequency f and amplitude A; initial phases are omitted

For a digital implementation (2.9) can be rewritten in a discrete time

$$u[n] = \sum_{k=0}^{M-1} A_k \cos\left(2\pi f_k \frac{n}{f_s} + \phi_k\right) \tag{2.10}$$

where f_s is the sampling frequency, and n is the integer time step index.

2.1.2.1. Evolution of Spectrum

A number of spectral components M of a synthesized signal can be much larger than the actual number of separate partials that need to be represented. Using densely distributed component frequencies formula (2.9) alone can lead to a spectrum that varies on a short time scale, where each evolving partial is produced by a group of interfering components. It is more common though that components directly represent distinct partials. Such spectral structure is easier to control and can be simply scaled when a different pitch is required, but it does not change over time. Time evolution of the spectrum is introduced through implementation of amplitude envelopes instead of constant amplitudes of interfering components. Similar operation can be applied to component frequencies (Fig. 2.9).

The signal with envelopes for both parameters can be written as

$$u(t) = \sum_{k=0}^{M-1} A_k(t) \cos\left(2\pi f_k(t) t + \phi_k\right) \tag{2.11}$$

where the envelopes $A_k(t)$ and $f_k(t)$ are slowly-varying control functions. In a discrete time (2.11) transforms into

$$u[n] = \sum_{k=0}^{M-1} A_k[n] \cos\left(2\pi f_k[n] \frac{n}{f_s} + \phi_k\right) \tag{2.12}$$

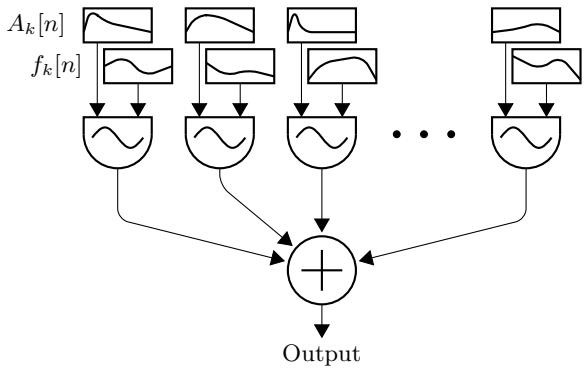


Figure 2.9. Additive synthesis with evolution of spectrum controlled separately for each oscillator through amplitude and frequency envelopes, $A_k[n]$ and $f_k[n]$, in a discrete time n; such configuration corresponds to formula (2.12)

2.1.2.2. Control Data

Additive synthesis may deal with a substantial amount of control data. It is assumed [470, 485], that a number of partials required for adequate approximation of a sound of an acoustic instrument lies between 32 and 64. It varies, however, depending on the instrument and pitch. Assuming that we only need to produce partials fitting within an auditory frequency range of a young, health human, then for pitch A4, usually tuned to f_0 between 440 and 443 Hz, we need at most 45 harmonics. The number shrinks to only 11 two octaves higher at pitch A6, but reaches 180 in case of A2, and could even exceed 700 for very low pitch A0 (usually the lowest key on a grand piano keyboard)[1]. In Figure 2.10 a total of 37 harmonics of oboe playing D5 can be observed. They are compared to 37 harmonics of tuba playing D1, which constitute only a part of its spectrum that fits below 1400 Hz. Lower pitches obviously require far more oscillators, even if we recognize that starting at some frequency, decreasing amplitudes would make higher partials imperceptible.

[1]Some of commercial additive synthesizers aiming at very detailed sound reproduction allow to combine hundreds of partials: 500 in Image-Line Harmor, 512 in AIR Loom, and 320 in Native Instruments Razor 1.5.

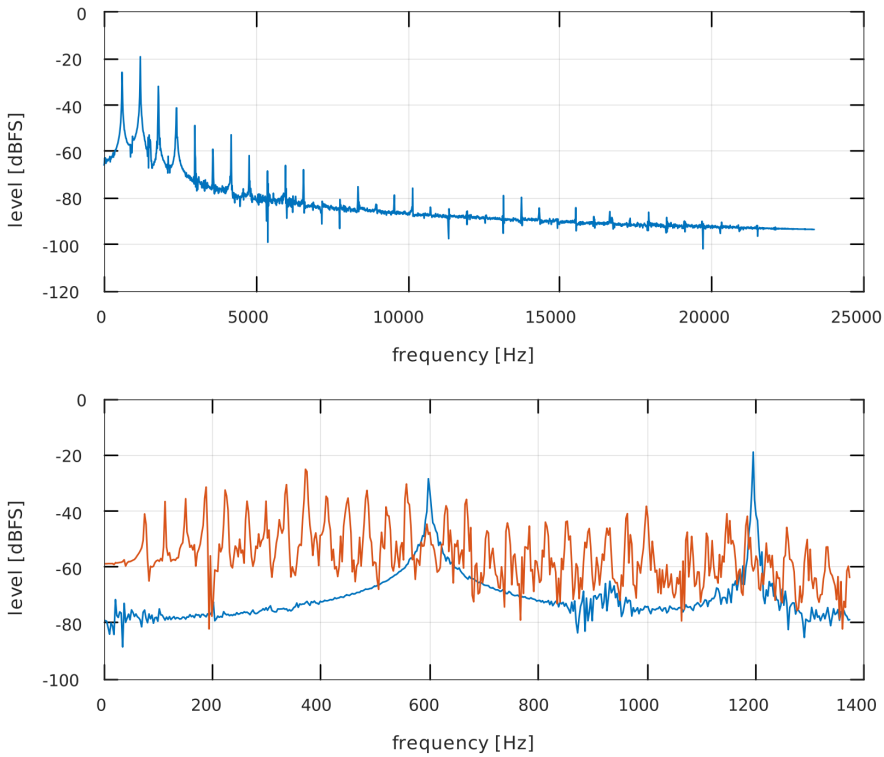


Figure 2.10. Comparison of a number of observable harmonics between an oboe playing pitch D5 (blue) and a tuba playing pitch D1 (red); top plot shows 37 oboe harmonics observable below 24 kHz; bottom plot displays 2 oboe and 37 tuba harmonics that fit below 1400 Hz

Considering that parameters of each component can change independently from the rest, it poses a serious control problem. A solution is either to automate most of control data operations, or to reduce amount of independent variables. A reduction is possible through approximation of slowly-varying control functions or by relating parameters of separate components.

Data Reduction

The largest amount of data is contained within envelopes. Therefore it is common practice to approximate both, amplitude and frequency envelopes by a piecewise linear functions – such technique is referred to as **line-segment approximation** [470]. Number of segments may vary to efficiently fit approximated envelope. Segments can be determined manually, usually with an aid of sound visualisation tools, but automatic procedures are also available [537]. According to Tolonen et al. [557], 10 segments used to approximate envelope lasting 500 ms of a signal with 44100 Hz sampling rate allow to reduce data by 69:1.

Further data reduction is possible if partials are in harmonic relation. With this assumption f_k can be supplemented with $(k+1)f_0$ and (2.11) changes to

$$u(t) = \sum_{k=0}^{M-1} A_k(t) \cos(2\pi(k+1)f_0(t)t + \phi_k) \tag{2.13}$$

or in discrete time

$$u[n] = \sum_{k=0}^{M-1} A_k[n] \cos\left(2\pi(k+1)f_0[n]\frac{n}{f_s} + \phi_k\right) \tag{2.14}$$

Higher component frequencies are related to the fundamental frequency which can either be a constant value f_0 or an envelope $f_0(t)$. If latter is the case, the same envelope will be scaled proportionally through all components, thus representing pitch deviation.

Even though it is possible to use common frequency envelope through all components, the same cannot be done with an amplitude envelope. It would be overly simplistic, considering that various parts of spectrum evolve differently. It is however possible to divide components into several groups belonging to sections of spectrum that evolve together. Oscillators within a group are controlled with a single, common amplitude envelope and a common frequency function. Such technique is called **control grouping** [485] or **group additive synthesis**, as presented with digital implementation by Kleczkowski [298]. Division into groups is based on the amplitude envelope criterion, where a measure of similarity can be a simple Euclidean measure of distance

$$\rho(Y_j, Y_k) = \sqrt{\sum_{n=1}^{N} (y_{jn} - y_{kn})^2} \tag{2.15}$$

where j and k are harmonic numbers of compared partials, n is the sample index, Y_j and Y_k are vectors storing all envelope values y_{jn} and y_{kn} of partials j and k, and N is the number of samples in envelope. Even though partials within a group have a common amplitude envelope, it is possible to scale it separately for each partial. Common frequency envelope for a group may be chosen arbitrarily from within envelopes of group partials. Alternatively, it may be calculated as a mean of all frequency functions, once they have been scaled down by a harmonic number. In case of cello and clarinet sounds grouping is difficult to perceive even with as few as three groups. For a trumpet four or five groups are required [298].

Sources of Control Data

Not only a substantial amount of synthesis data is a control problem, but providing its source is an equally important issue. Even with a modest configuration of 32 oscillators, a simple 4-segment ADSR envelope (Fig. 2.7c), and fixed component frequencies, one needs to set 160 values beforehand. It obviously points at adapting some kind of automated procedure for control data acquisition.

Roads [470] specifies five ways of obtaining control data, also referred to as **driving functions**. Some of them lead to new, and possibly abstract sounds, while others tend to recreate existing sounds.

The first way is to use data from entirely different domain and map it onto synthesis parameters. Such data might originate, for instance, from a shape of a natural object or a skyline, although any object or concept that can be modelled in either geometric, stochastic or otherwise mathematical way can be used. As an example, Cádiz and Ramos used a momentum distribution of a one-dimensional Gaussian quantum particle in an infinite square well [94]. The second way involves creating a composition algorithm that would generate driving functions according to some assumed constraints. Yet another way utilises higher-level musical concepts, such as phrases, or clouds. The fourth way relies on an intentional and manual use of the above methods combined in a creative way. The last way, which receives an increasing attention due to advances in methods of signal analysis, is the resynthesis.

2.1.2.3. Resynthesis

Resynthesis is a technique of synthesizing sounds in which synthesis parameters originate from the analysis of existing sounds [470, 300]. It is also commonly referred to as analysis-resynthesis or analysis-synthesis [61]. Its primary objective is to transform the original sound using capabilities of a synthesizer. The process has three stages, as depicted in Figure 2.11: analysis, editing, and synthesis. Analysis of a source sound has to provide parameters matching those required by the synthesizer, or at least such that can be converted accordingly. In editing stage parameters of the original sound are modified to change its chosen properties, such as pitch, duration, or timbre. Details of each stage can vary, since the technique can be applied not only to additive, but to other synthesis methods as well.

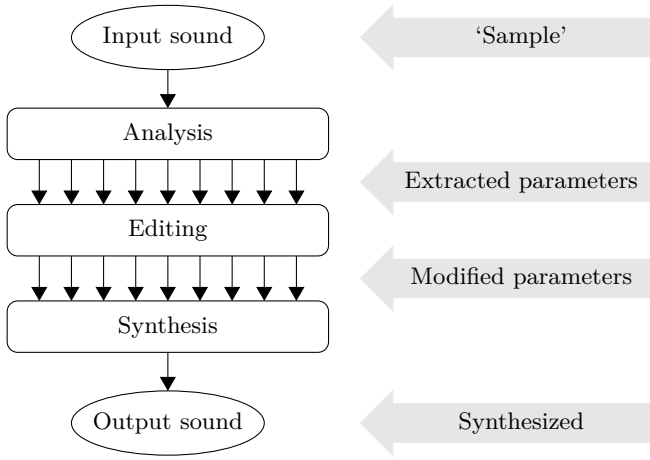


Figure 2.11. A resynthesis diagram; an existing sound (a *sample*) is analysed to extract parameters required for synthesis; parameters can be modified to introduce desired changes into the original sound; modified parameters are used to synthesize a new sound

In case of additive method the analysis has to produce parameters for envelopes of partials. Depending on the analysis method and parameters required by a synthesizer a set of extracted parameters may include either amplitude envelopes only, or more – amplitude and frequency envelopes, as well as phases. A number of analysis techniques is utilised, most of which are based on filter banks or numerical signal decomposition with fast Fourier transform (FFT) [138].

Vocoders are tools originating from speech analysis and resynthesis, that have been successfully adapted to a musical sound synthesis. They divide a signal into parallel channels associated with frequency bands. The most basic one, **channel vocoder**, keeps only an amplitude of channels. The **phase vocoder** (PV) [190, 449, 387, 214, 168, 300] improves the original channel vocoder by supplementing amplitude data with a phase information. PV has equally spaced frequency bands, therefore if immediate data is used alone, it is suitable for harmonic signals only. There are, however, PV extensions that allow to handle pitch changes such as *vibrato*, or signal inharmonocity, as shown by Moorer [387]. As of analysis technique, both – a bank of band-pass filters or a short-time spectrum analysers such as short-time Fourier transform (STFT) – can be utilised [168, 557].

While immediate data regarding a signal component obtained from PV consists of amplitudes and phases, **McAulay–Quatieri** algorithm (MQ) [358, 300] extracts all three types of information needed for full additive synthesis: amplitude, phase, and frequency. Such a triplet of parameters is estimated for each signal component at subsequent locations in time determined by the hop size of STFT, which is used in MQ to analyse subsequent signal windows [557]. Both, a hop size and a number of components, can vary in time to adapt to the analysed signal. In each signal window prominent peaks are searched for in the surroundings of integer multiplies of f_0. The range of search can be adjusted, but it is typically limited to $\left(k \pm \frac{1}{4}\right) f_0$, where k is an index of assumed harmonic partial. Peaks not exceeding noise threshold level are removed, and the remaining ones are connected by the peak continuation algorithm that produces amplitude and frequency trajectories for detected sinusoidal components. Components are not assumed to start and end together – their durations are independent.

Both methods, PV and MQ, are well suited for analysing signals consisting of a limited number of sinusoidal components. Signals containing noise are not analysed properly unless a very large number of components is assumed, and even then the result of such resynthesis usually differs significantly from the original sound [557]. This problem affects not only the analysis stage, but the synthesis stage as well, since additive method is very inefficient and in consequence inaccurate in synthesizing noise. As of frequency envelopes, they are estimated most accurately in case of slow and continuous changes. Such changes are typical for the majority of acoustic instruments, where they are caused by *vibrato*, or initial and final phase of bowing or blowing.

Other methods of analysis can also be applied, as long as they provide parameters required by a particular additive synthesis implementation. A choice of analysis type can also be based upon a transformation the signal is going to be subjected to in the editing stage. Roads [470] gives a list of such transformations. Pitch shifting without time scaling, and time scaling without pitch shifting allow to separately manipulate

two basic note parameters – pitch and duration. Other transformations are timbre-related or otherwise aimed at producing new sounds. They include various ways of manipulating frequency ratios of partials, creating spectral hybrids by replacing some envelopes with envelopes extracted from different sounds, interpolating from one instrument timbre to another, enhancing resonance regions, inverting spectra, stretching transient regions, and cross-synthesizing. Roads explains **cross-synthesis** as a technique of applying parameters extracted from one sound to synthesize other, with a possible change of their meaning. For instance, amplitude envelopes of one sound may control phases or frequencies of the second one.

Thus the resynthesis may be performed with two different objectives. The first one is to allow less constrained control over recorded sounds, such as independent change of pitch and duration, required to freely play a synthesizer. The second one is to utilise existing sounds for generating new, perhaps surprising and interesting ones, without resorting to arduous manual entry of overwhelming amount of parameters.

2.1.2.4. Control of Pitch, Duration, and Timbre

Control of Pitch

It is a common practice to store component frequencies f_k of the additive synthesis signal not as absolute values, but as ratios to the fundamental or the lowest frequency f_0. It can be expressed as the following modification to (2.11)

$$u(t) = \sum_{k=0}^{M-1} A_k(t) \cos\left(2\pi b_k(t) f_0 t + \phi_k\right) \tag{2.16}$$

where $b_k(t)$ are frequency envelopes related to f_0, or time-varying multipliers of f_0. Expression (2.13) is a special case of (2.16), where all partials vary in time according to a common envelope $f_0(t)$, and their frequency ratios are harmonic, that is $b_k = k+1$.

A component frequency structure in which all frequencies are related to a single value f_0 makes controlling a pitch a straightforward procedure – it involves changing f_0 only. All of the remaining f_k automatically follow, thereby the whole spectrum shifts with f_0.

The frequency shift does not affect sound duration nor contents of envelopes. It does, however, change the timbre of sound, because amplitude envelopes $A_k(t)$ are attributed to component indices rather than to absolute frequencies, so when the frequencies shift, formants shift as well. This is rather undesired outcome, and it can be partially solved by simulating registers, i.e. introducing dependence of A_k on f_0 ranges.

Control of Duration

In case of time-invariant additive synthesis, as in (2.9), duration can be controlled simply by running oscillators for as long as needed, or by gating their output using control events such as *note-on* and *note-off* MIDI messages. It is straightforward in time-evolving signals as well, assuming that envelopes are approximated by line-segments. One of the segments – synchronous in all components – needs to be identi-

fied as a sustain phase of sound evolution, during which envelopes remain fixed for as long as required. Other envelope segments are not altered, thus duration adjustments do not affect transients or fades.

If there is no sustain segment in envelope, one or more selected segments might be stretched according to required duration change. This, however, is problematic in case of real-time control, when target duration is not known beforehand. Moreover, it does not reflect behaviour of acoustic instruments and may sound unnatural.

Finally, in digital implementations envelopes might not be approximated, and instead could store detailed time-evolution of parameters in a discrete-time domain. Sampled envelopes can be resampled – either as a whole, or in selected sections only, leaving the rest, e.g. fast transients, unaffected. Such a selective procedure would require envelopes to be complemented with markers denoting parts excluded from resampling.

None of the duration control mechanisms presented affects pitch, nor spectral structure of a sound – it is one of advantages of the additive method. Due to this reason additive resynthesis is often utilised as a duration-control mechanism in sample-based methods, such as concatenative synthesis.

Control of Timbre

The working principle of additive method is essentially a design of a spectrum. Therefore a spectral aspect of timbre is precisely controlled, to the level of amplitude and phase of a single partial. Additive method can synthesize any discrete spectrum with partials arbitrarily evolving in time, controlled by envelopes. However, it is this detailed control which is also a weakness of the method: a change of timbre requires modifications of an extensive set of parameters. While it is arduous to modify the parameters individually and manually, two solutions can be applied to automate the procedure. The first is to use data sets prepared and stored beforehand. The second is to employ algorithms that generate larger sets of control data on the basis of much smaller collections of user-provided input parameters.

Ready to use data sets, or *presets*, can originate from any of previously mentioned sources, such as data of a different meaning (graphics, such as landscapes, geometry, physics), human invention, or analysis of existing sounds. Since only some combinations of values in extensive parameters space result in musically attractive sounds, in commercial applications presets store such potentially interesting sets provided by the manufacturer on the basis of his superior knowledge, understanding, and experience with the synthesizer. On the other hand, if the synthesizer has a full resynthesis capability, timbre of any recorded sound can be utilised as a pattern, with synthesis parameters obtained through analysis.

There are virtually no constraints on the algorithms employed to help managing large data sets in additive synthesis. In experimental implementations they are more inclined towards unconventional, unprecedented associations. In commercial solutions the goal is to provide easier, more intuitive control for the end user. That is why such algorithms are aimed at a convenient user interface, grouping of parameters, and creating additional abstract layers of control. They can provide extensive envelope design and modification facilities, modularise sound processing, allow mapping of

parameter groups onto various physical controllers, or translate high-level timbre-related concepts, such as articulation[2], onto actual envelope parameters.

Registers, Dynamics, and Articulation

In case of synthesizing sounds of acoustic instruments a notion of timbre separates into areas of registers[3], dynamics[4], and articulation. Playing range of a typical instrument is divided into a few registers (Fig. 2.12). In the additive synthesis a change of register is simulated by separate sets of control data – one for each register. A set is obtained through analysis of sound belonging to the appropriate register of an instrument, and is associated with an exclusive range of pitches produced by the synthesizer. While playing a synthesizer, a set of control data representing particular register is chosen, depending on the pitch played, which in case of a MIDI keyboard might simply be a key number.

A change of dynamics in a musical instrument has a twofold effect. It affects the amplitude of generated sound, but is responsible for a change of timbre as well. The latter is a result of a change in playing technique required to produce different levels of dynamics. A technique used to simulate dynamics-related timbre change is similar to the one used to simulate registers. Two or more sets of control data have to be acquired by analysing sounds of an instrument performing on different dynamics levels. These sets have to be associated with a counterpart of the dynamics in the synthesizer, which is commonly attributed to the MIDI *velocity* value.

The articulation is a more complex problem. In music it is not only associated with performance technique of separate notes, but also with note transitions. As for single-note articulations, such as *staccato*, its implementation follows the scheme of registers or dynamics. Distinct sets of control data are attributed to various performance techniques, and assigned to a controller of choice or to a note attribute in a sequencer.

The above-described solutions for simulating registers, dynamics, and single-note articulations in additive synthesis are not unlike the multisampling technique employed in sampling synthesis. The difference is that in sampling sound samples with different timbres are reproduced directly, while additive synthesis analyses them to acquire envelope parameters, though both synthesis methods use 'snapshots' of real timbres.

Note transitions however, require a different solution. The problem is particularly prominent in instruments controlled by breath or bow, while performing *legato* articulation, and has been widely researched [537, 191, 10]. Here additive synthesis has the upper hand over sampling. The advantage has two sources. The first is the line-segment approximation of envelopes that allows to selectively skip or connect envelope segments of adjacent notes. The second is a separate processing of partials, allowing to preserve their continuity in a transition phase. In the simplest case, en-

[2]In music, 'articulation' is the element that controls performance technique and note transitions.

[3]The term 'register' denotes a section of pitch range of a particular instrument that shares common timbre characteristics. Switching to a different register results in a change of timbre that is caused by e.g. a change of string in violin or by overblowing in woodwinds.

[4]In music, 'dynamics' refers not to a range of sound levels, but to one of the musical elements that controls the loudness of music and its changes through applying appropriate performance technique.

velope segments following the sustain phase are omitted in a preceding note, while in the following note segments preceding the sustain are omitted, and both sustain phases are connected, as presented in Figure 2.13.

If required, continuity or other desired behaviour of partials can be properly modelled by shaping their frequency and amplitude envelopes. Figure 2.14 shows two different examples of note transitions in a melody performed by a violin. Due to capability of simulating natural sound transitions additive synthesis is often used to connect notes in sample-based synthesis methods [504, 336, 505].

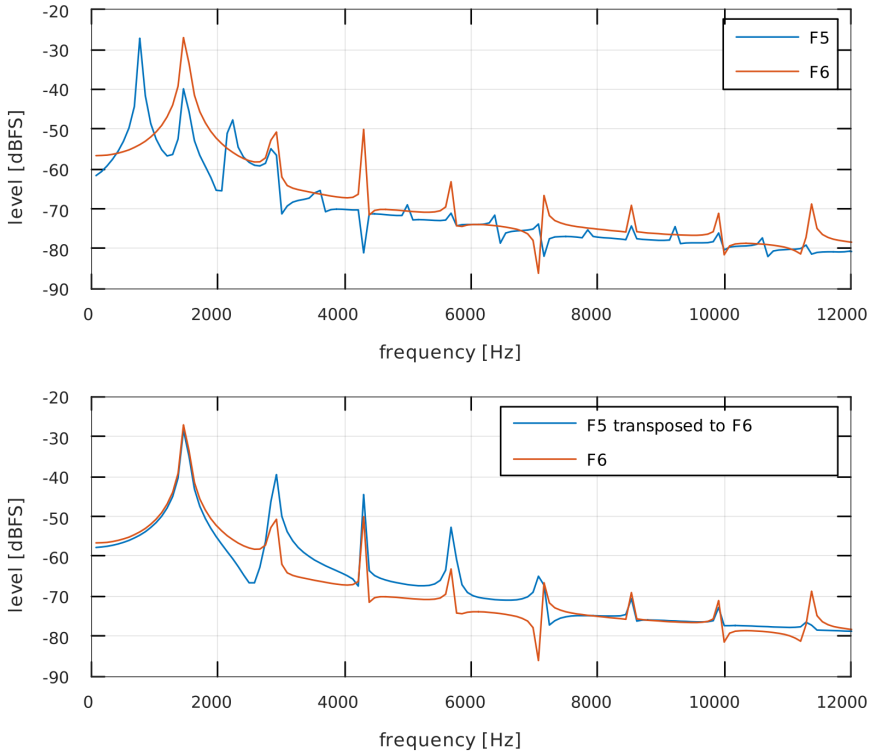


Figure 2.12. An illustration of a register-based spectrum difference; two flute sounds have been analysed; the lower one, pitched F5, belongs to the middle flute register; the higher one (F6) belongs to the high register; analysis results were used to carry out additive synthesis of two sounds presented in the top plot; thereafter synthesized F5 has been transposed one octave up by keeping amplitudes of partials, while doubling their frequencies; bottom plot compares spectra of a sound generated on the basis of analysis of a correct register (F6), with a sound transposed to a higher register (F5 to F6); as a result of transposition partials 2–4 are noticeably amplified (by approximately 10 dB), and partial 8 is similarly attenuated, which makes the sound unnatural

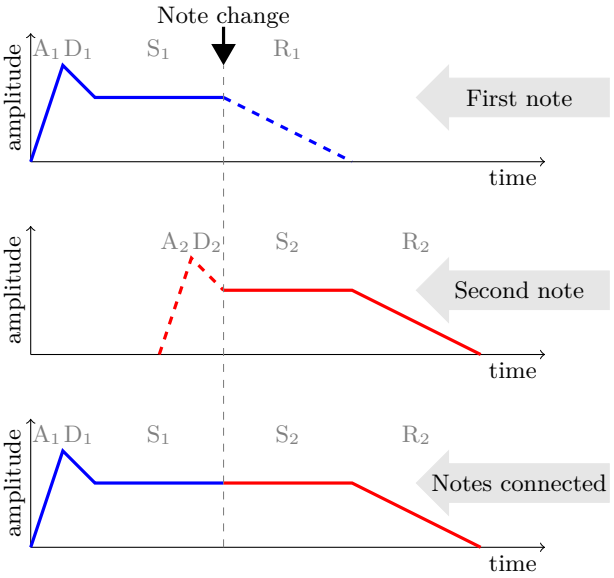


Figure 2.13. Connecting ADSR amplitude envelopes of partials belonging to adjacent notes performed *legato*; the first note begins normally (e.g. start of a bow movement), with initial segments present (A_1 and D_1) and continues to the sustain segment (S_1); the second note starts before the first finishes, but it jumps to the sustain phase (S_2) without going through A_2 and D_2, because it is not started separately (e.g. the bow continues its movement, only a pitch has changed); R_1 segment is skipped as well, because the instrument still plays; if there are no more notes to play, then R_2 follows S_2 and the sound finishes

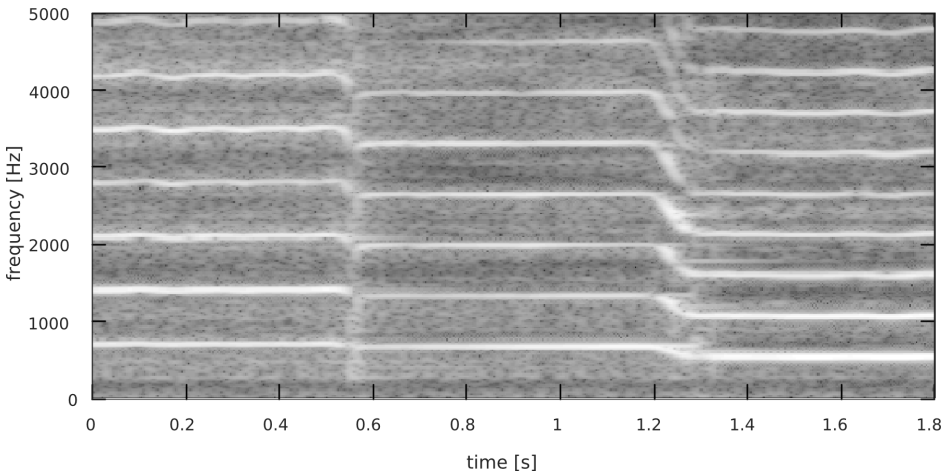


Figure 2.14. Two subsequent pitch transitions in recording of a violin; the first one is performed in *détaché* articulation, with a change of bow direction, the second one is *legato* with slight *portamento*, on one bow

2.1.2.5. Variants of Additive Synthesis

The additive method is based on a general principle of combining simple partials into a complex sound. It allows to synthesize virtually any sound, providing that sufficient numbers of partials have been combined, and that their envelopes have been modelled with adequate accuracy. While being powerful, the method has its drawbacks: it is computationally demanding, requires a sizeable set of parameters, and does not produce good noise components. These issues have been addressed by methods that are either variants of additive synthesis, or were – at some point – based on it.

The main source of inefficiency is a parallel operation of a multitude of oscillators. Many solutions of the problem have been proposed, such as **multirate additive synthesis**. The technique, presented by Phillips and Purvis [435], uses *a priori* knowledge of component frequencies and applies multirate DSP techniques to adapt oscillators frequency ranges.

The inefficiency can also be addressed by switching from an oscillator-based signal generation to a calculation of inverse Fourier transforms of consecutive windows. The method is referred to as **inverse FFT synthesis**, or FFT^{-1} synthesis, and was proposed by Rodet and Depalle [474]. It groups all signal components into a series of spectral envelopes and stores them in a series of STFT frames. IFFTs of frames are calculated and added with overlapping, thus producing output signal [557]. Two different windowing functions are used in time and frequency domains: a triangular one results from linear interpolation of amplitude and frequency between frames, and the second one needs to have low sidelobes due to efficiency reasons [474]. Inverse FFT synthesis can generate periodic, and quasi-periodic signals. It can also synthesize noise components of arbitrary characteristics.

A different way of addressing lack of capability to produce noise components is **spectral modelling synthesis** (SMS) presented by Serra and Smith [511, 512]. SMS is a resynthesis technique that decomposes a signal into deterministic and stochastic components [557]. The former is tone-based and can be obtained using MQ or similar algorithm, while the latter is noise-based. After the deterministic component is resynthesized, it is subtracted from the original signal in time or in frequency domain. The result – the residual signal – is used to form the stochastic component. For this purpose a bank of filters controlled by a series of spectral envelopes is applied to shape a white noise signal [470]. A convenient side-effect of separate processing of both components is a possibility to exclude stochastic component from pitch shifting operations. While SMS can adequately model sounds of many instruments, very short transients tend to be spread in time domain. It is a consequence of dismissing phase information of the original signal. Only magnitude spectra of the stochastic component are shaped, while the phase spectra are random.

Due to transients issue, another method has benn developed as an expansion of SMS. **Transient modelling synthesis** proposed by Verma et al. [595] separates the residual signal into steady noisy components and transients. The first stage of analysis corresponds to the SMS analysis. The resultant residual signal, in TMS referred to as the first residual, consists both – noise and transients. TMS detects the transients

and fits them with a parametric model. Transients are synthesized and subtracted from the first residual, creating the second residual that contains slowly evolving noise only. Since sinusoidal model cannot adequately represent transients, pulse-like signals are first transformed to a frequency domain to become sinusoidal. In order to obtain not complex, but real-valued frequency-domain representation, the discrete cosine transform (DCT) is used instead of the discrete Fourier transform (DFT). Parametric model of transients is thus obtained by sinusoidal modelling (DFT) of DCT of the actual transients. A DCT block must be sufficiently large for overlapping DFT windows – it was determined [595] that one second is adequate.

Fractal additive syntehsis (FAS) is a method of additive resynthesis, similar to SMS, presented by Polotti and Evangelista [448]. It is best suited for voiced sounds. Like in SMS, a signal is decomposed into deterministic and stochastic component, but the process is based on harmonic-band wavelet transform. Unlike the original SMS, FAS method preserves attack transients. Polotti later proposed a pitch-synchronous extension of FAS via time-varying cosine modulated filter banks [447]. The extension allows to synthesize a wider class of voiced sounds, including those with variable pitch, such as *vibrato* or *glissando* effects.

An interesting effect of a timbre evolution can be obtained with **spectral interpolation synthesis** (SIS), as shown by Serra et al. [510]. The evolution is achieved through interpolation between successive spectra using ramp functions. A sequence of spectra is controlled by a *spectral path*. The path connects amplitude values representing matching harmonics in subsequent spectra. It is defined by these values and a mixing function, e.g. a linear one. A set of spectral paths resemble a set of piecewise-linear approximated amplitude envelopes. In SIS though, break points of all envelopes are simultaneous. The auditory effect might be similar to crossfading between sound samples, however due to the additive synthesis approach SIS allows for a better control over a generated signal, such as independent time stretching and pitch shifting.

Yet another modification to the additive synthesis is the **Walsh function synthesis** [470]. It is one of several additive approaches that consider different than sinusoidal component signals. A Walsh–Hadamard transform can decompose a signal into components represented by rectangular Walsh functions [259]. A finite series of such functions can approximate an arbitrary periodic function. Instead of frequencies, the parameters representing components are *sequencies* that correspond to the one-half of the average number of zero crossings per second. Using Walsh function synthesis it is easier to produce rectangular-like signals that would otherwise require a large number of sinusoids. Though precise sine-shaped signals would require a large number of Walsh functions. One of the drawbacks of the method is that Walsh function components do not directly represent signal harmonics, and as such do not have immediate auditory representation, which makes the process of signal construction less intuitive.

2.1.2.6. Implementation Remarks

Additive synthesis is one of the oldest synthesis methods. It had been utilised even before a synthesis has been defined and established as a way of sound production. Its

principle was implemented in Cahill's Telharmonium on a turn of XIX-th and XX-th century to create various timbres for musical purposes [97]. But if a requirement for components to be sinusoidal is loosened, then the same principle has been utilised in music for centuries. Most notable example is the organ mixture, which is a mechanism that allows one key of a keyboard to control not one, but a group of pipes from different ranks. Through selecting different mixtures one can produce various timbres. The method is flexible and general enough to be implemented in a very wide class of synthesizer types, from electro-mechanical, through electronic, both analogue and digital, to purely software-based.

The last class gradually gains a dominant position. Software can rely on a computer hardware that is constantly improving, or at least, is improving faster than dedicated synthesizer hardware. Therefore most of commercial additive synthesizers are, as of now, software-based – usually in a form of a plug-in for a digital audio workstation (DAW) software [617].

Many of commercial software engines are not limited to additive method, but implement a wider set of synthesis methods. It allows to combine their elements, such as filters from subtractive method, recordings from sampling, or a modulation. Although filters may seem redundant in additive method, where the spectrum is controlled in detail, they are applied to the output signal for a convenience – they are faster and easier to set. In a combination of additive and subtractive method harmonic partials may be supplemented with either generated or sampled noise, subjected to filtration. Sampled signals might be used as a source for analysis and resynthesis. Vocoders (with as much as 34 channels) are popular as analysis method. Since storage is not a problem any more it is possible to extract practically unlimited number of spectral snapshots to create detailed envelopes.

Additive synthesizers are supplemented with effects ranging from frequency modulation for introducing dissonance into the harmonic structure, to spatial effects. Due to a principle of additive method, design of a complex, evolving echo is possible. Reverberation can be applied not only to the output signal, but also per partial, following pitch changes. The same applies to panorama effects – partials can have individual panorama settings.

The main emphasis is put on control issues though, which normally is a weak spot of additive synthesis and in commercial applications it would discourage inexperienced users. Envelopes can be modified in many creative ways, including graphical, or through grouping into formants. Some programs allow to use images, either loaded from files or drawn at run time, as a source of partial envelopes – the idea has been exploited in music for some time. Several parallel signals may be generated and mixed or morphed (crossfaded), which is a concept borrowed from vector synthesis. To tackle a large amount of synthesis parameters, or to quickly test various timbres, parameters might be randomised within predefined boundaries. Some programs display a modular structure, allowing a user to modify the sound production process. It is a standard to include presets or *patches* with factory-set parameters.

Current commercial software synthesizers use hundreds of partials. While it may seem to be a large number, it can be further increased to allow very detailed sound control. Savioja, Välimäki, and Smith showed, that it is possible to compute over

one million unique sine waves in real-time [490]. It can be carried out using graphics processing unit (GPU) available in 2010. Konopko [306], Kobayashi [303], and Crespi [142] further studied the subject. GPUs are massively parallel, and due to recent dynamic progress of game industry, they evolve rapidly. Considering a single precision floating point data type (FP32), top class GPU of 2010 had an estimated peak processing power of approximately 1.35 TFLOPS, while as of 2018 this value has reached 16.3 TFLOPS. This definitely solves the problem of processing power requirement, and the additive method can now be used practically without any computational constraints.

2.1.3. Subtractive Synthesis

A sound spectrum does not have to be produced in a laborious process of additive synthesis that starts from zero and adds subsequent sine components. An opposite approach may be applied as well. The procedure begins with a signal that is spectrally rich, and transforms it to match target spectrum by subtracting necessary amount from its components. Hence the method is referred to as the subtractive synthesis [470, 485, 61]. It would not be significantly different from the additive method if subtractions were performed on a partial basis. In subtractive synthesis, however, a precision of spectral shaping is traded for a convenience of control – the signal spectrum is shaped through filtering, thus a few parameters can affect large parts of the spectrum. Due to this principle, schematically presented in Figure 2.15, the method is also referred to as a source-filter synthesis [557].

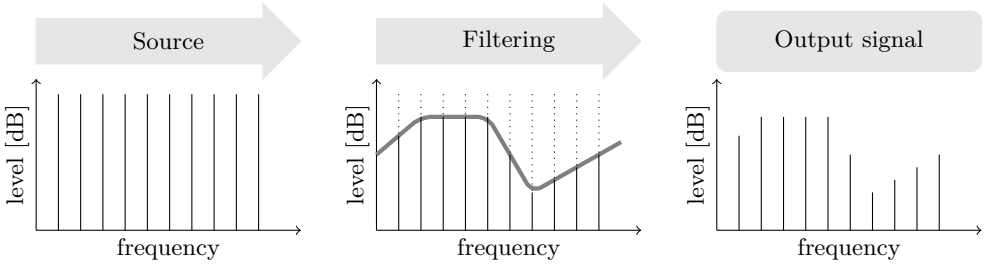


Figure 2.15. A general principle of the subtractive synthesis; the sound generation starts with an initial signal that has a rich spectrum; the spectrum is shaped in a desired way by subtracting its unwanted parts through filtering

The method was a particularly popular way of sound generation in commercial analogue synthesizers, with such notably examples as Buchla and Moog instruments. At least part of the acclaim may be attributed to the sound control mechanism. While in general a user is not able to control a single partial, the sound can still be intuitively shaped in spectral domain. And even though filter parameters require some understanding, it can be easily obtained through auditory experience – alteration of each parameter produces audible result.

A principle of filtering spectrally rich source signal is – in a way – an analogy to the process of sound generation in acoustic instruments. Although the parallel is only a crude one, since source-filter model disregards influence of the excitation and resonator coupling on characteristics of both [61], it is helpful for a musician. Through filtering one can introduce and control a phenomenon of formants [191, 105] – spectral peaks related with major instrument resonances. Thus a control of filter parameters translates into an intuitive control of instrument qualities.

2.1.3.1. Source-Modifier Principle

Sound production mechanism applied in the subtractive synthesis can be reduced to a source-modifier principle. The source is a signal generator, while the modifier is some configuration of filters and amplifiers. Therefore the following three elements can be controlled in the process of sound production:

- source waveform,
- filters parameters,
- and amplitude.

Selected parameters may be linked to physical controls of a synthesizer, such as sliders, knobs, joysticks, or pedals, to allow alteration of their values in real-time, as a part of performance. Such mechanism would be less practical in additive synthesizers with way too many detailed parameters to control in real time, but in case of subtractive synthesizers, with a relatively limited set of globally-operating parameters, it expands their expressive capabilities.

Signal Sources

Two classes of acoustic signals are used as a source of an initial signal in subtractive synthesis. The first is stochastic, and the second is periodic. The most basic stochastic signal is a white noise. It requires no parameters and in a linear frequency domain has a flat magnitude spectrum, therefore it is an excellent basis for filtering. Stochastic signal can be used either as a noise component required in sound of instruments such as flute or violin, or – shaped by a very short amplitude envelope – as a source of clicks or cracks produced at the very beginning of an excitation phase when plucking, striking, or playing sharp *staccato*. Noise can also be a base of some percussive sounds.

Signals from the second class – periodic – are produced by the VCO, and include waveshapes such as sine, triangle, square, sawtooth (Fig. 2.1), and pulse (Fig. 2.2). Periodic signals share a common auditory feature: they elicit a sensation of pitch [299], and therefore always require at least one parameter: a frequency. To be more precise, the actual parameter is a fundamental frequency f_0, since apart from sine all periodic signals have more than one partial, and all these partials are in integer frequency relations to f_0. Selection of a source waveform determines the initial spectral properties of a signal, including spectral envelope and harmonics content (all or only odd), that is later to be shaped by filters. Some waveforms expose additional parameters, such as pulse width, that allows to alter the initial spectrum in a continuous way.

Filters

Due to various kinds of filters applied in subtractive synthesizers, their parameters vary, although it is common that VCF has at least one parameter related to frequency, for instance a centre or a cut-off frequency, while others control the shape of filter characteristics. Amplitude and filter parameters may either by controlled directly, or through EGs and LFOs. Both of the latter introduce additional parameters that control temporal variability of a signal.

In subtractive synthesis one feature of filters plays the most important role: it is their ability to attenuate or boost selected areas of a signal spectrum. Therefore the paramount filter quality is its frequency response [470], and specifically magnitude frequency response. While most filters used for subtractive synthesis purposes are linear and time-invariant (LTI)[5], some effects applied during synthesis may be regarded as filters that are non-linear (e.g. dynamic compression) or time-varying (e.g. amplitude modulation) [525]. LTI filters do not introduce new spectral components to the signal, which makes them suitable for subtractive purposes – they only modify previously existing partials or spectrum bands.

Subtractive synthesis uses the following types of frequency response curves: low-pass, high-pass, band-pass, and notch (band-reject), as well as shelving filters [470]. If a filter provides more than one type of response curve it is referred to as a **multimode filter**. For synthesis purposes filters are controlled by a very limited parameter set with a flexible naming scheme. Particularly in digital implementations or virtual synthesizers parameter names originating from analogue and digital domains are being mixed. Usually one parameter controls position of the curve on a frequency axis and other one or two – its overall shape. In low-pass and high-pass filters the curves are characterised by a cut-off frequency and a slope parameter, which in musical applications is expressed in dB/octave. In band-pass or band-reject filters curve position is controlled by a centre frequency value, and shape – through a quality factor (Q) or a parameter referred to as a resonance, accompanied with gain. Parameters may differ depending on implementation. In many synthesizers the shape of low-pass or high-pass filters cannot be controlled, while in some it is controlled through a resonance parameter, as in band filters. For band-pass filters Q relates centre frequency and bandwidth [470]

$$Q = \frac{f_c}{f_2 - f_1} \tag{2.17}$$

where f_c is the centre frequency, f_1 is the low, and f_2 the high cut-off frequency (at –3 dB).

Rarely in subtractive synthesizers a quality factor is termed as such, and a resonance term is used instead. It refers to the auditory effect of accentuating centre frequency as a pass-band gets narrower. It is similar to the resonance effect in acoustic instruments, and hence the name. Resonance can be controlled not only in band-pass

[5] If time-variance of a filter is below audio rate, then for audio purposes such filter may still be treated as time-invariant. For a digital filter its coefficients should be practically constant at least over a duration of its impulse response [525].

and band-reject filters, where it peaks central frequency, but also in some low-pass or high-pass filters, where a peak appears near the cut-off frequency.

A band-pass or a notch filter is referred to as a **constant Q filter** if it adjusts its bandwidth to centre frequency in order to maintain constant quality factor. Constant Q filter spans over a fixed musical interval, and it maintains the shape of its frequency response in a logarithmic frequency scale. This feature may be useful with pitched sounds: if a centre frequency is related by a constant ratio to a fundamental frequency of filtered signal, then even if a pitch is changed, passband will always contain partials with the same harmonic indices – spectral structure will shift with pitch. Similar principle can be applied to low-pass or high-pass filters: their cut-off frequency can be related to the fundamental frequency of filtered signal. In commercial applications the feature is termed **filter scaling**, **pitch tracking**, or **pitch following** [485].

If a subtractive synthesizer is used for a resynthesis purpose it may include a number of parallel filters organised in a **filter bank** [470], usually operating as an equalizer. Such arrangement allows to shape spectrum of a signal in a flexible way, hence it can be referred to as a **spectrum shaper**. The level of detail however, does not reach capabilities of additive synthesis, where amplitude and phase of each partial is controlled separately.

The auditory effect of a low-pass filter is related to the attribute of sound brightness: lowering cut-off frequency makes the sound darker through lowering value of spectral centroid [498]. Shifting cut-off frequency of a high-pass filter also affects brightness – the higher, the brighter – but in this case the effect may also lead to pitch ambiguity, since fundamental frequency, and higher harmonics afterwards, will gradually disappear. Typical usage of a high-pass filter is to slightly attenuate fundamental frequency in relation to higher harmonics [485], which is characteristic for spectra of many acoustic instruments.

Even though filters embedded in subtractive synthesizers share a very similar control interface with a few simple parameters, their working principle and characteristics can actually differ significantly. Analogue solutions such as Moog filters [383] or their later improvements are often regarded as exemplary, due to more distinct features than basic digital implementations. Therefore contemporary digital subtractive synthesis aims at recreating these characteristics and features, as well as features of all elements of analogue synthesizers, through a technique of **virtual analog** (VA). Välimäki and Huovilainen claim [573] that VA allows to produce 'retro' sounds with modern computers. Digital simulations of analogue circuitry pose a number of issues, from aliasing to nonlinearity. Some possible solutions with a source code are presented in the work of Välimäki et al. [570]. A generalisation of the wave digital filter (WDF) theory with implementations and analyses of nonlinear devices, such as Moog ladder filter or Buchla lowpass gate, can be found in a thesis of D'Angelo [149]. Work of Zavalishin [615], aimed at digital musical instrument and effect developers, brings up not only emulations of existing analogue devices, but also guidelines how to design new VA digital filters. While VA is mainly applied in subtractive synthesis, it may also be regarded as a special case of physical modelling synthesis [61] where not mechanical, but electronic systems are modelled.

2.1.3.2. Synthesizer Designs

Unlike a clear arrangement of data paths in additive synthesis, a source-modifier principle is very general. Elements and control sources of a subtractive synthesizer, represented by unit generators, may be arranged in various ways. Thus the data flow is not fixed, and depends on particular implementation. This freedom of configuration is exploited in modular synthesizers.

Unit generators can be arranged as required, depending on the qualities of sounds that are to be produced, or assumed control scheme. Some examples of typical arrangements are presented in Figure 2.16. Arrangement (a) is suitable for analysis-resynthesis tasks. Input signal is divided into several frequency bands with band-pass filters. Each of bands is processed through EG that controls its gain. Amplitude envelopes can be obtained from analysis of a sample sound through a channel vocoder.

Configuration (b) uses two oscillators, possibly with different waveshapes. Both signals are processed in the same way: they are low-pass filtered and amplified. Both, VCF and VCA are controlled through separate envelopes. Fundamental frequency of the second VCO can be detuned, which gives an opportunity to produce inharmonic spectrum or beating. Here only two signals are mixed, but often three or more are used.

Arrangement (c) has a simple signal path, but introduces two effects due to usage of LFOs, as shown in Figures 2.4 and 2.5. The first LFO controls frequency, which produces *vibrato* effect. The second one controls gain, which results in *tremolo*. A filter here has two separate envelopes – one controls cut-off frequency of a low-pass filter, and the other one – its slope. This arrangement can be regarded as an improvement of a single path of arrangement (b). Several settings from (c) can be connected in parallel in a manner similar to (b).

In modular synthesizers a number of universal EGs may be assigned to any controllable destination, including amplitude through VCA gain, various VCF parameters, such as filter cut-off or centre frequency and filter resonance, VCO frequency, or – more often – VCO frequency deviation, LFO frequency, as well as other parameters mentioned in Table 2.2.

Regardless of chosen synthesizer design subtractive method demonstrates a primacy of control ease over detail by limiting amount of control data. In accordance to this principle EGs tend to be controlled through a very constrained parameter set, unlike EGs in additive synthesizers that may be composed of a relatively large number of segments.

Patch

Simple analogue subtractive synthesizers are usually limited to a single configuration of unit generators, such as one of presented in Figure 2.16, and to pre-defined effects, like these in Figure 2.4. On the other hand, full-featured modular analogue synthesizers allow almost free configuration of unit generators and controllers. Such configuration is customary referred to as a *patch*, because essentially it is an arrangement of patch cords that connect units. In analogue synthesizers there may not be a distinction between control and signal data, since both are just analogue volt-

ages. Patch cords that allow to connect any output to any input, beyond assignments shown in Table 2.2, certainly provide great freedom in experimenting and inventing new sounds and effects, though they obviously require caution.

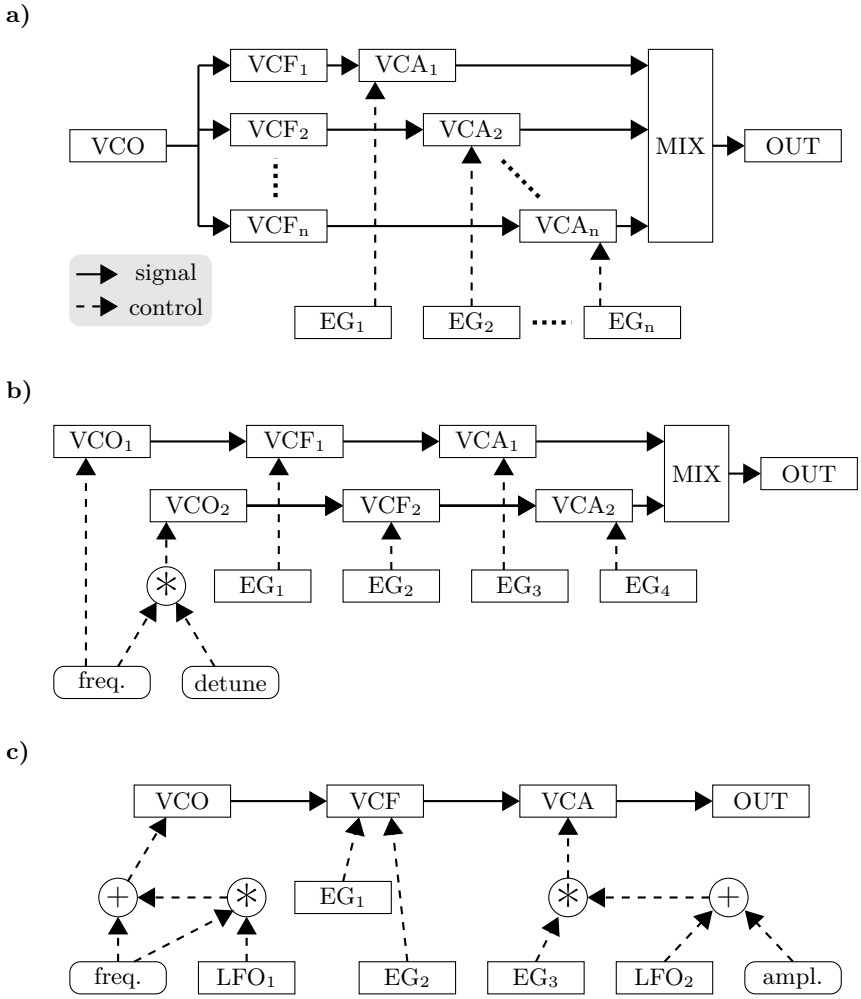


Figure 2.16. Examples of unit generators arrangements; in configuration (a) VCFs work as a filter bank – a set of band-pass filters – that divides signal spectrum; each of separate bands is amplified by VCA controlled through individual EG; in (b) signals from two separate oscillators are processed in parallel through low-pass filters and amplifiers before mixing; amplitude in VCA and cut-off frequency in VCF are controlled by separate EGs; VCO_2 frequency is set in relation to frequency of VCO_1 – they can be detuned to produce beating or inharmonicity; configuration (c) is the simplest, but introduces two LFOs, LFO_1 controls *vibrato*, LFO_2 controls *tremolo* effect; in low-pass VCF, EG_1 controls cut-off frequency and EG_2 – resonance or slope

A concept of patch – units connected with patch-cords – is present in many of contemporary graphical music programming languages. However, most of digital implementations, either in software or hardware form, make a clarification about a type of data sent: different connections are devoted for signal transmission and for control data.

Table 2.2. Examples of sources and destinations in modular subtractive synthesizers

Source	Comment	Destination	Comment
Pitch bend Modulation wheel Foot controller Breath controller Pressure Note number Note-on velocity Note-off velocity	Physical controllers	Rate Pan Spread Depth	LFO parameters
		Pitch Portamento time	VCO parameters
		Frequency Resonance	VCF parameters
		Envelope depth Panorama	VCA parameters
X-Axis Y-Axis Z-Axis	3-D controller	Noise level	—
LFO	Bipolar output, or Unipolar output	Attack Decay Sustain Release Attack curve Decay curve Sustain curve Release curve	EG parameters
VCA envelope VCF envelope Auxiliary envelope	Various envelopes		
Control sequencer	Programmable		
Voice number	—		

Modulation Matrix

Separation of control data allowed to form another concept. Pre-wired subtractive synthesizers implemented a simplified variant of modularity, referred to as **modulation matrix**. The name is not entirely established and various synthesizer vendors introduce their own variants such as: alternate modulation system, matrix control, routing, or mod matrix. A word 'modulation' in this context is very general and refers to controlling one unit (either a signal generator or its modifier) using output of other unit.

Modulation matrix is a set of source-destination pairs. A synthesizer has a fixed number of possible sources of control data, a fixed set of control data receivers, and a certain number of connections that can be established – channels. A connection is set by pairing a chosen source with selected destination, and setting a parameter that controls amount of modulation, usually referred to as *depth*, as shown in Figure 2.17.

Depth can be either positive or negative. One source can be used multiple times, to control several destinations. It is also common that depth of a particular modulation is controlled by another source, set in a separate modulation. For instance, LFO can modulate VCO frequency (one channel), while depth of this modulation is controlled by a physical controller such as modulation wheel (another channel). Such chains can follow multiple-levels deep.

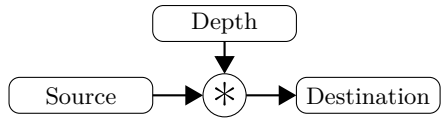


Figure 2.17. An element of a modulation matrix; *depth* is a user-set parameter that controls the amount of modulation; it can be either positive or negative

Depending on particular synthesizer, modulation matrix can be implemented and presented in various forms (Fig. 2.18), from a simple one-column arrangement with limited number of channels, to a full matrix that allows to connect every source to every destination simultaneously. Examples of common sources and destinations are presented in Table 2.2.

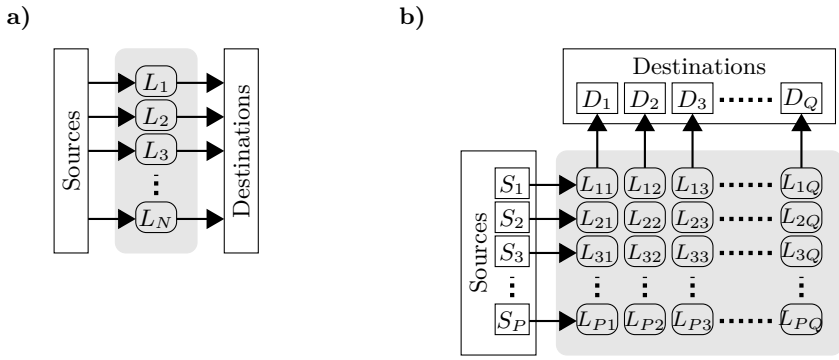


Figure 2.18. Forms of a modulation matrix (L_i or L_{ij} represents modulation depth): a) N-channel, one-column; b) $P \times Q$ element matrix; in (a) up to N source-destination pairs can be defined from among a larger set of sources and destinations; in (b) all P sources can be connected to all Q destinations

2.1.3.3. Resynthesis

Additive and subtractive synthesis share a similar sound production principle. Both shape a spectrum directly, and both deal with time-variability of a signal through implementation of envelopes. Therefore, as is the case of additive synthesis, subtractive synthesis is also suitable for resynthesis technique. A general approach is the same, however subtractive and additive methods have different limitations, and hence details vary.

Vocoder

Subtractive resynthesis is aimed at creating such filter or configuration of filters, and controlling them with such driving functions, that provided with properly chosen excitation signal they reproduce a source sound allowing to alter its pitch, duration, or timbre. First applications of subtractive resynthesis regarded the speech synthesis area. Musical applications followed later. The first device that performed subtractive analysis and resynthesis was the **vocoder** [175, 177, 176, 496, 189, 470]. The analysis stage in analogue vocoder is performed by a set of fixed-frequency band-pass filters that are fed with the original speech signal. Their outputs are directed to envelope detectors that generate driving functions in a form of voltages proportional to the amount of energy in analysed frequency bands. Resynthesis stage utilises a subtractive synthesizer in an arrangement similar to one presented in Figure 2.16a. Synthesizer uses bank of band-pass filters identical to those from the analysis stage, fed with wide-band excitation signal, e.g. pulse train or noise. Band-pass filtered excitation signal is directed to VCAs controlled by driving functions from envelope detectors. Output of all VCAs is mixed to produce output signal. The overall process is schematically shown in Figure 2.19.

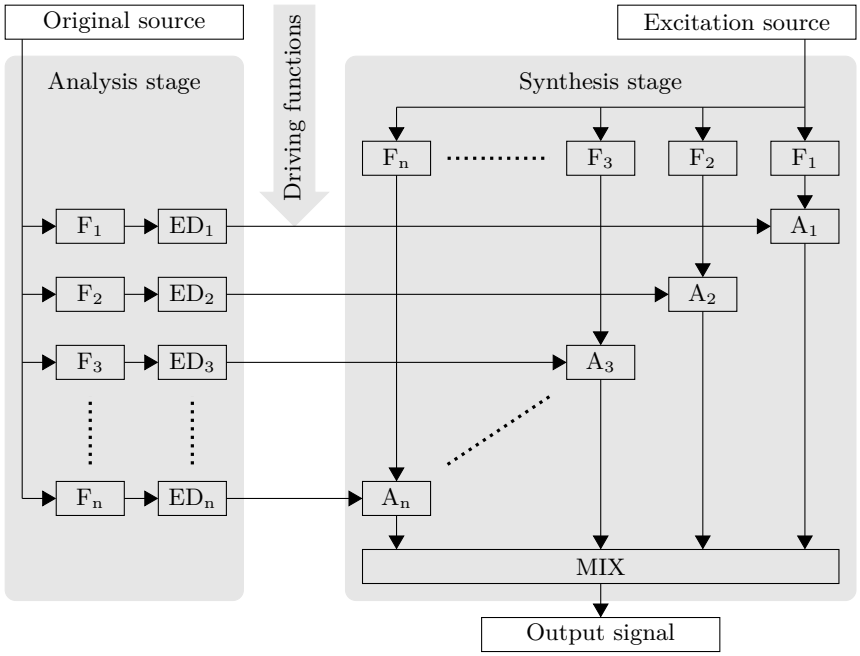


Figure 2.19. Vocoder diagram; symbols: F – band-pass filter, ED – envelope detector, A – amplifier (VCA)

Originally the source signal was a human voice, and band-pass filters were used to extract its formant structure in a form of time-varying spectral envelopes. Vocoder

superimposes this structure onto excitation signal. Excitation should have a rich, and possibly flat spectrum, but it does not have to be abstract or artificial. Therefore, referring to an excitation-resonance musical instrument model, vocoder separates resonance, in a form of driving functions, from excitation, and substitutes original excitation with any signal of choice.

An initial goal of the process was to reduce amount of data required for speech transmission or speech synthesis. Only driving functions were transmitted, while the excitation was not. On the analysis side it was possible to detect if the sound was voiced or unvoiced, and switch excitation signal on the synthesis side accordingly – to a periodic signal or noise.

Separation of resonance and excitation proved to be useful for musical purposes. Slow-varying driving functions are carriers of duration-related data, such as rhythm and tempo, as well as some of source spectral characteristics. The excitation signal is a source of fundamental frequency, and in consequence – pitch. Therefore, by separate control over properties of both, excitation and source signal, or by modification of driving functions such as time-stretching, it is possible to control pitch and time structure separately, or apply spectral characteristics of one sound to the other.

An interesting and popular vocoder effect is a 'singing instrument'. It is produced by using instrument sound as excitation source, and driving it with data from analysed speech signal. A usual setting is a keyboard-controlled synthesizer connected as an excitation signal, and microphone connected as a source (Fig. 2.20). The effect is achieved by simultaneous playing the synthesizer and speaking to the microphone. In this kind of setting the excitation can be referred to as a carrier, and the source – a modulator. Actually two separate synthesizers are at work here: the keyboard-controlled one (any synthesis method), and the second one (subtractive) in vocoder.

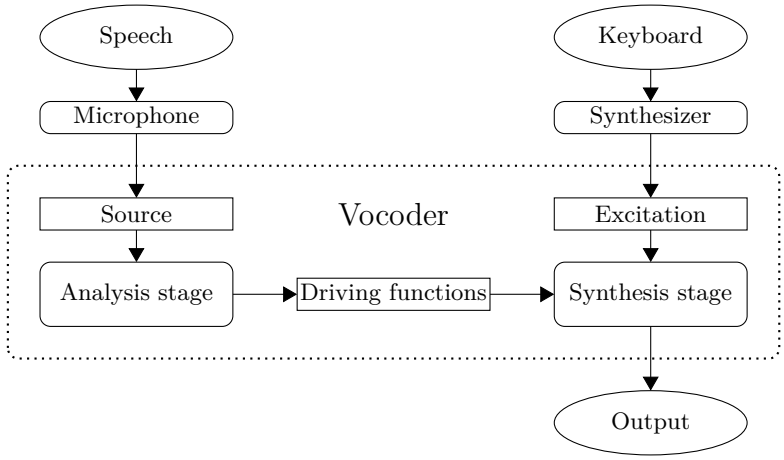


Figure 2.20. A 'singing instrument' vocoder setup; microphone signal is a source of driving functions, while keyboard-controlled synthesizer provides an excitation signal; playing the synthesizer while speaking to the microphone produces the effect of 'singing' spoken text with pitch and timbre of the synthesizer

A vocoder can work with as little as eight octave-wide frequency bands. Enhancing spectral resolution by increasing number of filters in a filter bank improves resynthesis fidelity, but only to a certain point, after which further increase of filters number does not bring extra improvement. The issue has to be addressed by applying a more elaborate method.

Linear Predictive Coding

While the analogue vocoder may be the most straightforward approach to the subtractive resynthesis, other possibilities exist. A more advanced approach is represented by the linear predictive coding (LPC). LPC attempts to estimate an all-pole filter with a magnitude frequency response that matches analysed signal [189, 350, 557]. The filter, designed using the optimisation procedure, is applied to a synthetic excitation signal, and produces the best approximation of a signal that has been analysed. Such approach is well suited for signals displaying spectral peaks, or formants, which is a case of human voice and some musical instruments. In phase vocoder analysed frequency bands are equally spaced, which works well with harmonic spectra only. Similarly, in STFT magnitudes and phases are determined in equally spaced points, and to make a good spectrum estimation STFT requires many such points. Contrary to foregoing two, LPC is a parametric method for determining spectrum envelope. Therefore it does not assume harmonicity. As a result LPC can be applied to a broader set of cases.

LPC may be considered a data reduction technique. Compared to a raw waveform, LPC reduces data by analysing a signal in a series of time-frames, or blocks of samples, typically between 50 and 200 per second [470]. Each frame consists of a set of data sufficient to reproduce a segment of the original signal. A common set includes the length of a frame, the average amplitude of the original signal, the average amplitude of the estimated output produced by the inverse filter, the pitch of a frame, and the coefficients of the all-pole filter. Pitch can be determined using any f_0 detection algorithm, while frame amplitudes may be estimated by calculating their RMS values. A series of snapshots of filter coefficients from subsequent frames contains information about spectral evolution of the analysed signal.

Unlike basic vocoders, LPC analyses the signal further. Each frame is categorised either as voiced or unvoiced. This affects a choice of excitation signal in the synthesis stage (Fig. 2.21). Voiced frames are resynthesized from a pitched pulse train. For unvoiced – excitation is a white noise. Decision whether a frame is voiced or unvoiced involves applying one of several heuristics. It can be based on the f_0 estimation error, using the amplitude ratio of the estimated filter output to the original signal. Unvoiced frames produce large errors, so setting proper error threshold can be an initial hint.

An all-pole filter that contains formant structure of the analysed signal is determined in a two-step procedure described in detail by Makhoul [350]. Firstly, predictor estimates an inverse filter. The process involves a linear prediction of the next sample in a time series through linear combination of prior samples [136]

$$\hat{u}[n+1] = \sum_{i=0}^{m} a_i u[n-i] \tag{2.18}$$

where $\hat{u}[n+1]$ is the estimate of $u[n+1]$. A common method to calculate coefficients a_i involves the autocorrelation function [136]. The inverse filter is an all-zero FIR filter that applied to the original signal attempts to produce the excitation signal. Since the original excitation is not known, a synthetic is assumed: either a pitched pulse train or a white noise. A difference between the assumed excitation and the excitation obtained through application of the all-zero filter to the analysed signal is the error of prediction [136]

$$\epsilon[n+1] = \hat{u}[n+1] - u[n+1] \tag{2.19}$$

Estimation performed by LPC is optimal with a minimal mean square error (MSE) criterion applied over the entire frame containing N signal samples [136]

$$MSE = \frac{1}{N} \sum_{n=0}^{N-1} \left(\hat{u}[n+1] - u[n+1]\right)^2 \tag{2.20}$$

Minimisation results in the best fit of the inverse filter. In the second step the inverse filter is inverted, thus producing the all-pole IIR filter, which was the initial objective.

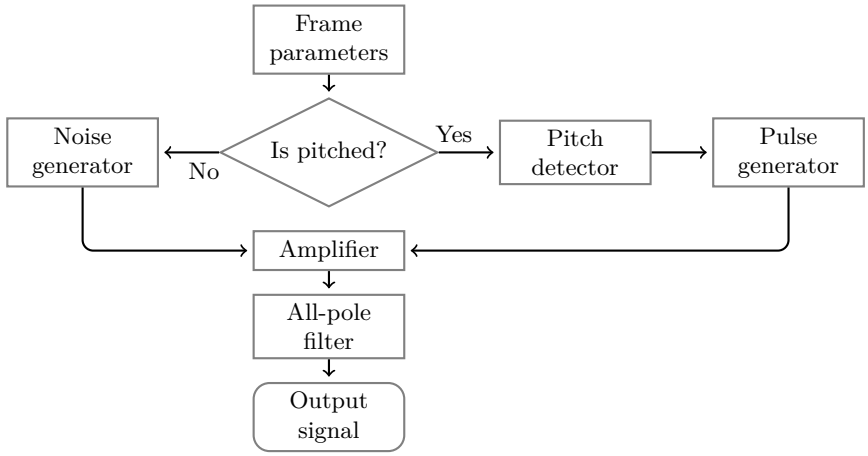


Figure 2.21. A general principle of the LPC synthesis
Source: author's elaboration, based on Miranda [373]

LPC viewed as an analysis and resynthesis tool is, in fact, a kind of vocoder, and it was developed with similar aims: to reduce data amount in speech transmission and synthesis [355]. And, similarly to vocoder, musical applications followed [98, 388, 319, 317, 167, 318]. If LPC is utilised for speech processing, twelve-pole filter is considered sufficient, but in musical synthesis acceptable quality involves much larger numbers – more than 55 [388].

Simple vocoders and LPC share a common musical advantage: a capability to separate excitation, along with its pitch and spectral structure, from modulation signal with its information on spectral evolution. They can 'blend' properties of

different sounds, such as speech and instrument, which is a form of cross-synthesis. Moreover, LPC provides extensive editing capabilities due to its frame-based data set. According to Roads [470] frame duration can be changed, as well as order of frames, or particular frame parameters, which allows to introduce effects such as pitch *glissando*, *crescendo*, or *trill* – by manipulation of f_0 or frame amplitude. Through such manipulation a speech can be changed to singing.

Even though LPC provides an interesting musical capabilities and pushes synthesis quality beyond that of simpler resynthesis techniques, the quality cannot be improved over certain point [388]. Attempts to make the output signal less artificial involve using more complex excitation: a **multipulse cluster** instead of a simple pulse train [21]. Contents of such cluster are based on the frame analysis.

2.1.3.4. Control of Pitch, Duration, and Timbre

Control of Pitch

Due to separation of excitation and modification in subtractive synthesis, pitch is controlled independently from spectral envelope and its temporal evolution. It is determined solely by excitation signal fundamental frequency. There is a possibility to alter pitch perception not through the oscillator, but through signal modifiers. It involves either a filter removing some number of lower harmonics along with a fundamental frequency from excitation signal, or applying modulation effects with modulation frequency high enough to produce audible sidebands. Both effects however are rather incidental – they can make a pitch ambiguous, but do not allow to freely control it. Therefore a normal way of controlling pitch is through VCO frequency parameter.

Unlike additive synthesis that can, but do not have to link frequencies of all partials, and allows their independent control e.g. to introduce inharmonicity, pitched oscillators in subtractive synthesizers produce periodic signals with all partial frequencies in harmonic relation. It is not possible to control frequency of a single partial. Even after filtering periodic signal through a bank of band-pass filters to separate partials, only modification available for a single partial is related to amplitude, and not frequency. If inharmonicity is required, it can be produced either through introducing sidebands with modulation effects or through mixing a number of oscillator signals with detuned fundamental frequencies – the latter is a more common solution.

Control of Duration

Duration of a sound produced through subtractive synthesis is controlled independently from pitch or timbre. If an evolution of amplitude needs to be imposed, it is carried out by applying EG to control signal gain over time. Depending on a type of envelope (Fig. 2.7) two cases are possible. If the envelope has a *sustain* segment (Tab. 2.1) duration can be extended as required. Otherwise, the amplitude will eventually fall to zero, and note will end. If a *note-off* event is sent before the envelope reaches its last *release* segment, it starts the last ramp-down from current value towards zero with a slope of *release* segment. These properties and behaviour are inherited by other synthesis methods, such as wavetable, that share a source-filter or, more generally, a source-modification model.

Control of Timbre

While it is commonly assumed that subtractive synthesis separates pitch and timbre control, actually it is not a complete separation. The reason is that a spectrum of produced sound has two origins. The first is the original spectrum of a source signal, and the second – the filter applied. A spectrum of pitched source can be almost flat, as in simplified Figure 2.15, if the signal is a pulse train with a very small duty cycle. More often though, a spectrum has some kind of decaying envelope – either monotonic (Fig. 2.1), or not (Fig. 2.2). This spectral structure follows changes of oscillator frequency, and it cannot be prevented. The filter part, however, can behave as required: filters cut-off or centre frequencies can follow pitch or can have fixed values. In the first case the whole spectrum, its source and filter part, shifts with pitch. It produces a sound that always has the same number of partials in the same proportions. In the second case source part shifts, but filter part does not. This is not unlike a behaviour of some acoustic instruments, where a fixed spectral structure, originating from resonances of instrument parts, usually in a form of a number of peaks, is imposed on a floating excitation spectrum (Fig. 2.22).

Unlike additive method, which is limited to constructing spectrum from discrete, sinusoidal partials, subtractive synthesis can mix periodic, pitched signals with filtered white noise, and is therefore capable of producing continuous or partially continuous spectra. Such approach usually lacks precision, but is efficient in synthesizing percussive sounds, as well as breath or bow noise.

A timbre depends not only on spectrum, but also on spectral evolution. In subtractive synthesis it is controlled by EGs as well as LFOs. Another part that affects timbre is configuration of unit generators, or settings in modulation matrix – routing and depth. Therefore a complete timbre settings involve:

- choosing an oscillator waveform,
- arranging unit generators,
- and setting parameters of signal modifiers.

Registers, Dynamics, and Articulation

Subtractive and additive methods have similar capabilities to simulate timbre-related features of acoustic instruments, with a few exceptions. Due to a performance-oriented, less precise control mechanism, subtractive method can reproduce fewer details of instrument features, though it is able to reproduce noise components. Considering similarities, both methods can handle register, dynamics, or articulation changes by using multiple sets of timbre-controlling parameters per instrument. When either a register is switched, a dynamics is changed, or a different articulation is selected, a set of parameters associated with a particular situation is applied.

In case of closely connected notes such as in *legato* articulation, subtractive method can simulate smooth pitch transitions, although they are less convincing than additive approach. While additive synthesis is able to precisely control each partial in transition region according to prior analysis of a real instrument behaviour, subtractive method simply skips parts of envelope segments that are responsible for prior release and following attack phase, with an optional pitch *glissando*, applied equally

to all partials. The result is smooth, yet not entirely realistic. Alternatively, a more advanced resynthesis technique can be applied.

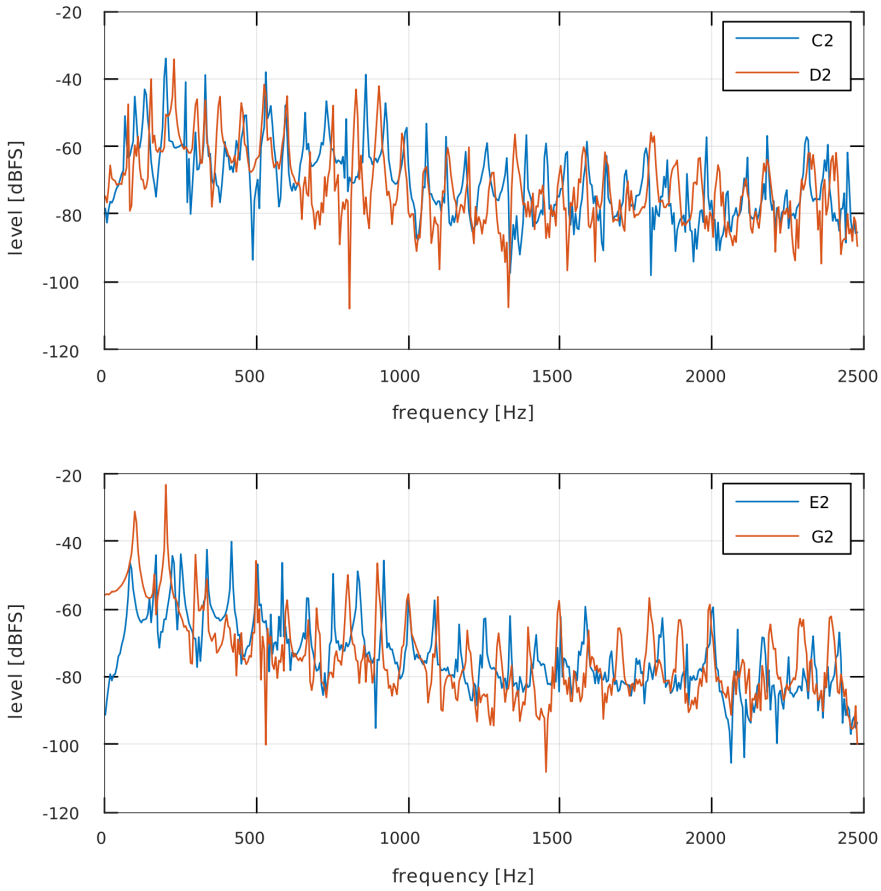


Figure 2.22. Magnitude spectra of four cello notes: C2 and D2 (top plot), E2 and G2 (bottom plot) recorded in anechoic chamber; apart from harmonic partials decaying with frequency, a number of broader peak regions can be observed, most notably around 200 Hz, and below 900 Hz; the lower can be attributed to C_3 corpus resonance [92], the higher – to the first bridge mode [191]

The **diphone synthesis** is a method designed for speech synthesis [430, 431, 414, 499] that has been later applied to music. The concept, initially implemented as a subtractive analysis and resynthesis technique, is based on assumption that in speech stable sounds are interchanged with transition sounds. Similar approach can be applied to sounds of musical instruments. Musical diphone synthesis utilises a dictionary of stable and transition sounds [470]. Diphones from a dictionary, described by pitch and loudness, are concatenated to produce arbitrary output. Yet concatenations could lead to audible discontinuities. In order to avoid them, models based

on transition rules are applied [537, 475]. A diphone is divided into interpolation and non-interpolation zones that are treated differently. Non-interpolation zones are preserved since they represent rapid signal changes that should not be altered. Interpolation zones can be stretched or compressed using LPC method – they are connected through transition zones. Manipulation of interpolation zones allows to produce arbitrary output.

Fundamentals of diphone synthesis were later applied in other sound synthesis methods, and its variants are known under a number of different names, such as concatenative synthesis [452, 513, 504, 505, 344], reconstructive phrase modelling [336], musical mosaicing [619, 124, 123], or phrase assembling [446, 445].

2.1.3.5. Descendants of Subtractive Synthesis

A source-filter or a more general source-modifier sound generation principle proved to be a very robust approach. Even though a basic subtractive synthesis is less capable in very detailed reproduction of acoustic instruments than the additive synthesis, it makes up for this deficiency with convenient and reasonably intuitive control schemes, and powerful reconfiguration ability of modular synthesizers. Both, strengths and flaws together decided that the subtractive method branched into a multitude of new – often quite successful – synthesis methods.

One of directions was to replace a few basic source waveforms with a much larger set in the **wavetable** synthesis. New waveforms became available with the introduction of digital oscillators and increasing storage space. Gradually, timbre-related operations had been transferred from modifiers to source, with the assist of sophisticated waveform selection mechanisms, while filters started to play a secondary role. The method ceased to be spectral, and became waveform-based. Sampling took the process even further. Obviously, basic sampling is not a source-filter method. It was soon however, that a lack of signal modification capabilities brought subtractive modifiers to the sampling method in a technique referred to by Russ as **sampling & synthesis**, in contrast to a basic 'sample replay' method [485].

A second way involved development of improved modifiers. Filters were often used to reproduce spectral effects of physical phenomena taking place while playing acoustic instruments. Mapping these effects to appropriate instrument parameters, e.g. in case of using wind or string instrument-like synthesizer controllers, required a model. Thus source-modifier approach met physical modelling, and its traces can now be observed throughout various **physical modelling synthesis** methods. These traces are particularly explicit in simplified methods, such as **modal synthesis**, **MSW** (McIntyre, Schumacher, and Woodhouse) [360] or **Karplus–Strong** method [286].

Lastly, a number of methods and tools picked selected elements of source-modifier approach and arranged them in a different manner. **Modulation methods** built upon a principle of modulating – which was initially an effect only – to produce different spectra. Still, a source-modifier principle holds, even though complex relations between multiple sources and modulators in **FM** (frequency modulation) algorithms differ from a straightforward layout of a pure subtractive synthesis. Some variants of **additive synthesis** implemented filtered noise as a component sound alongside sinusoidal partials. A traces of subtractive method can be observed even in **granular**

synthesis, when grains contents are processed through envelopes to obtain desired spectral characteristics. Finally, the principle of sources, modifiers, and modularity is a basis of many sound programming languages.

2.2. Waveform-Based Methods

2.2.1. Wavetable Synthesis

A wavetable synthesis is a group of digital and hybrid sound synthesis methods that operate according to the source-modifier principle [485]. Its core is a digital signal generator based on a table-lookup mechanism (Fig. 2.23). Hence the method can be also referred to as a table-lookup synthesis [470]. Signal from generator, either before or after conversion to analogue domain, may be modified by filters, amplifiers, LFOs, and EGs.

Table-lookup signal generator requires two elements: memory, and digital to analogue converter (DAC). The method assumes that a signal generated is periodic, therefore it is necessary for the memory to store a single period of a waveform only. Memory is organised as a one-dimensional array, where subsequent signal values – signal samples – are stored in a time-order. Thus a wavetable can be considered a sound sample truncated to a single period. The array is read cyclically, and values read are sent to DAC, or undergo further processing if the implementation is purely digital.

Table lookup operation can be written as

$$u[n] = tab[\phi_n] \tag{2.21}$$

where ϕ_n is the integer phase index produced by a modulo counter. Phase index is incremented according to the following expression

$$\phi_n = (\phi_{n-1} + \Delta\phi) \bmod L \tag{2.22}$$

where 'mod' is the *modulo* operation, $\Delta\phi$ is the value of increment, and L is the array length.

Wavetable is actually a simple, efficient, and hence very common implementation of a digital sinusoidal oscillator [61]. Contrary to analogue VCOs, it is not limited to simple waveshapes only, but can produce any periodic signal, since data array can store an arbitrary set of values. Contents of the array have no impact on the oscillator efficiency – it is always the same array read operation. Efficiency and simplicity are the reasons why a wavetable is very often a basis of other digital synthesis methods.

Signal Frequency

Phase index increment is related to a signal frequency by the following formula

$$\Delta\phi = \frac{Lf}{f_s} \tag{2.23}$$

where f is the signal frequency, f_s is the sampling frequency, and L is the length of a wavetable.

According to (2.23) signal frequency can be controlled not only through $\Delta\phi$, but through L or f_s as well. These however, are more problematic. New array length requires all signal samples to be either recalculated, or read from another data storage. New sampling frequency is more viable if signal is going to be mixed with other signals in analogue domain. In case of digital mixing it is still possible, but summing signals with different sampling frequencies involves their prior resampling to a common value. Therefore the most convenient way to change signal frequency is to change the phase index increment.

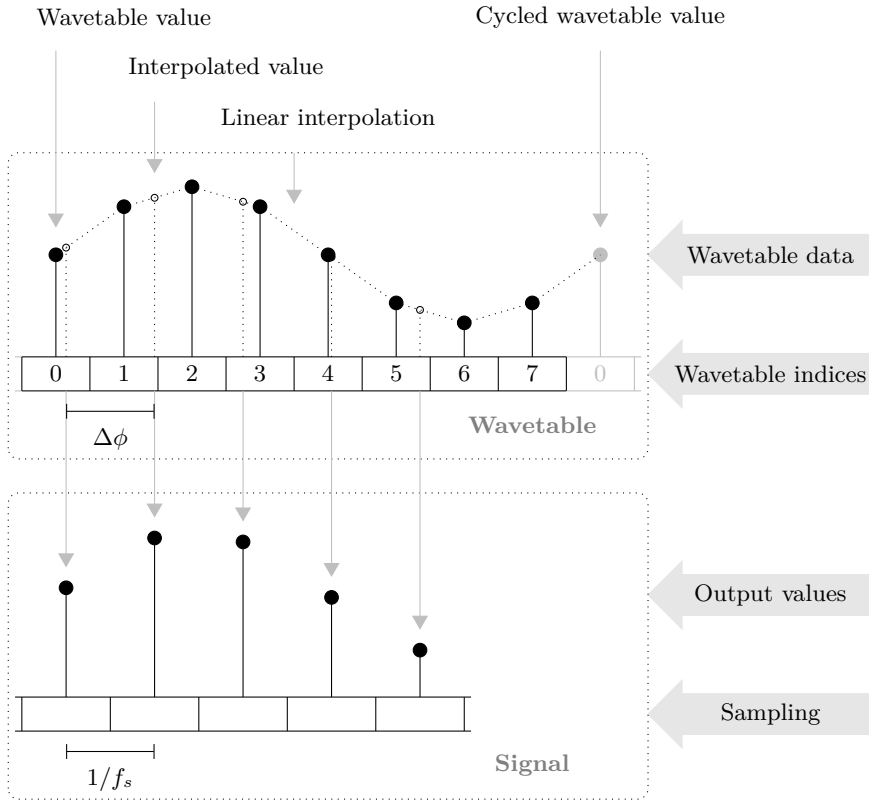


Figure 2.23. Wavetable synthesis with linear interpolation; wavetable containing L sample values is read periodically with sampling rate f_s, and phase index increment $\Delta\phi$ set according to a target frequency, as in (2.23); if $\Delta\phi$, and consequently wavetable index, has a fractional part, there is no immediate value available and interpolation of some sort has to be implemented

Indices of array cells are integer numbers, therefore according to (2.23) it might seem that only a limited number of discrete frequencies is possible, with other values unattainable. In fact, it is possible to produce any frequency using interpolation, which may be considered resampling of a wavetable. The easiest to implement is

the nearest-neighbour method, however, it has the worst auditory effect, producing audible table-lookup noise [470]. The noise can be reduced either by use of larger array, or by applying linear (Fig. 2.23) or higher-order interpolation. Even linear interpolation brings a very significant improvement over nearest-neighbour approach. According to Moore [386], for a wavetable storing 1024 values of sine function, SNR is no worse than 109 dB if the linear interpolation is applied, compared to 48 dB for the nearest neighbour method. Relations between table length, interpolation order, spectral roll-off of stored signal, and SNR were studied by Dannenberg [150].

2.2.1.1. Single-Cycle and Multi-Cycle Wavetable

Single-Cycle

A single-cycle wavetable, or a wavecycle [485] is the most basic case of wavetable synthesis, and the closest to subtractive method. It can also be referred to as a single-cycle oscillator, when only a generator, and not the whole synthesizer is the object of interest. Single-cycle oscillators can be implemented in various synthesis methods as signal sources.

Like analogue VCO, single-cycle oscillator produces a fixed waveform, which is either processed digitally, or – in hybrid synthesizers – converted to analogue signal and further processed in this form. Processing involves low-pass filtering, usually in a pitch-following mode.

The wavecycle oscillator and the VCO differ in the available waveshapes – the wavecycle allows to choose among a larger set, or to define own waveshapes. In early appliances entry of table values was carried out through sliders and slider-scanning circuits, but soon more advanced interfaces followed, such as graphics monitor with a light pen [485].

Larger number of waveforms available allows to implement 'velocity-switching'. Depending on key velocity value a different waveform is chosen, altering timbre in an attempt to simulate change of articulation that occurs in parallel to a change of dynamics. 'Velocity-switching' has a similar effect to linking velocity control and filter parameters, though the former can be a source of more complex changes.

Multi-Cycle

Both, the multi-cycle and the single-cycle oscillators allow to produce various waveforms. The difference is, that in a single-cycle a single waveform is chosen at a time, while in a multi-cycle various waveforms are produced in sequence. Therefore an additional control mechanism is introduced – a choice and order of waveforms within a sequence.

Reproducing waveforms in a sequence has two interesting auditory effects. Firstly, spectral elements of all component waveforms are present. Secondly, signal period is lengthened, multiplied by a sequence length, thus a pitch drops [485]. If sequence length is a power of two, pitch drops by a number of octaves. Otherwise – by a different interval (Fig. 2.24). This sequence-related pitch sensation can cease to be perceivable if a sequence is long enough to shift it below the auditory range. As a way of producing

more complex signal with a limited memory, waveforms in sequences can be time-reversed, inverted, or both at the same time. Longer, more varied sequences are better at producing noise-like output. Very long sequences are, in fact, nearing a sound sample concept.

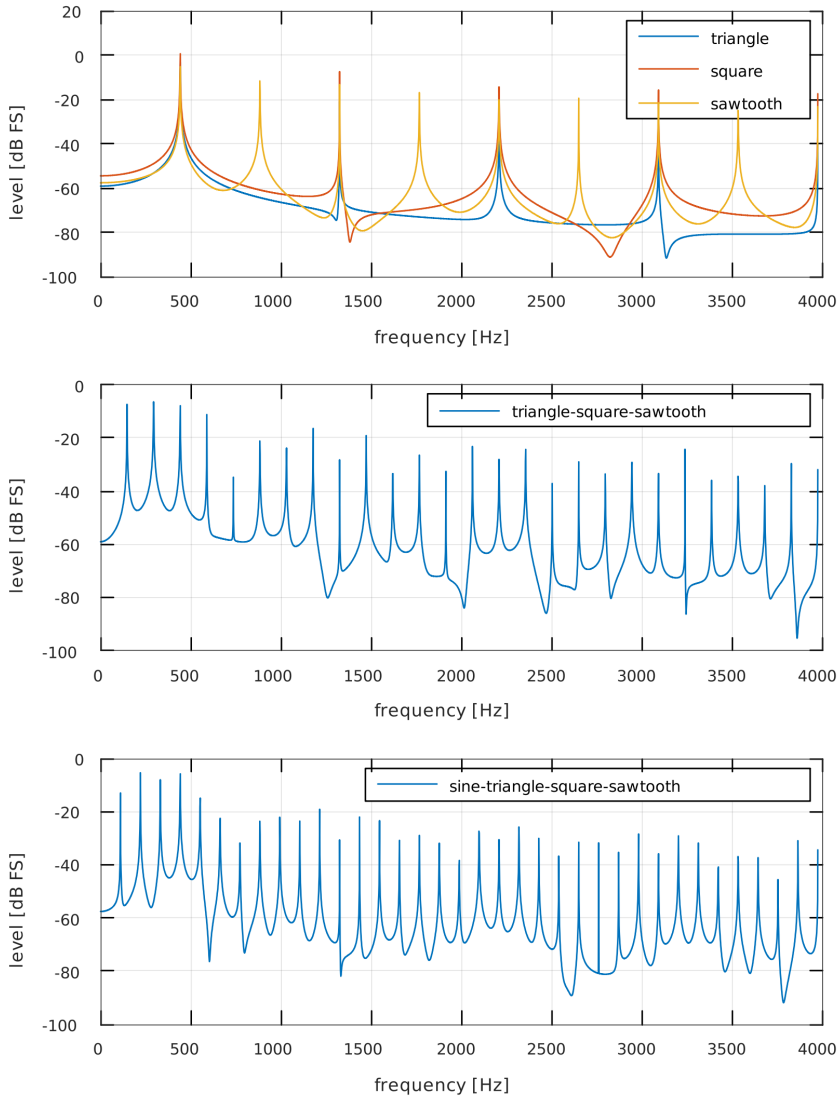


Figure 2.24. Magnitude spectra of three- and four-component multicycle oscillator signals; component waveforms spectra are presented in the top plot ($f_0 = 441$ Hz); a three-component sequence in the middle plot has an effective f_0 shifted 3× below f_0 of its components, and four-component sequence in the bottom plot – 4×

2.2.1.2. Signal Modification and Evolution

In general principle, wavetable does not differ from subtractive synthesis: a spectrally rich source signal is shaped through filters and amplifiers with aid of EGs and LFOs. However, wavetable synthesis mixes purpose of a source and a modifier. Much of signal shaping capabilities are moved to the signal source, while the role of filters shrinks in the majority of implementations, except the most advanced digital synthesizers [485]. It is common, that only a simple, low-pass resonant filter is available.

Effects of filters can be reproduced in a much simpler way, at the expense of memory requirements, through introducing waveforms characterised by a spectrum that is already filtered in a desired way. Arranging such waveforms in sequences with gradually changing spectra can simulate filters controlled by envelopes. This technique requires large numbers of waveforms with very similar spectra, otherwise waveform switching produces audible 'steps'.

Wavetable Access

The term 'wavetable' is often used as a general name for a broad class of methods that produce signal by reading its samples from memory – from single-cycle oscillators, through multi-cycle synthesizers, to samplers. In commercial applications they are often difficult to clearly categorise, when e.g. a synthesizer itself has full wavetable capabilities, with a set of signal modifiers and flexible access to sample memory, but in factory default settings it uses simple sample-replay capabilities only.

Single-cycle oscillators can switch waveforms on demand, but they do not switch them in sequence, which is a distinguishing feature of multi-cycle oscillators. A fully-fledged wavetable method builds upon a multi-cycle synthesis, which simply repeats a fixed sequence, by adding the ability to modify the sequence on a cycle basis. The other way around, single-cycle and multi-cycle techniques are simply feature-constrained variants of a full wavetable method. Sampling may behave like a wavetable method, but it handles control and processing of long recordings differently. It is therefore considered a separate method, though it has its roots in the wavetable synthesis.

That being said, a subdivision of wavetable techniques is based mostly on a way of accessing and handling sample memory. Wavetables store separate signal periods. These periods may represent subsequent stages of signal evolution. If they are reproduced in sequence, it can be considered a sound sample, with a notable difference: wavetables can be picked and used separately, while a sound sample in a conventional sampler is not segmented into periods, therefore there is no direct way to select and use one.

Wavetable synthesis uses two storage areas. The first one contains cycles – single periods of waveforms. It is not modified while playing, and can be a read-only memory. The second one stores a sequence of cycles. A fully-fledged wavetable synthesis allows a sequence data to be accessed and modified during operation. Changes are carried out between waveforms, so once a waveform has started, it cannot be changed.

A sequence is accessed in one of two ways [485] (Fig. 2.25):

- **sweep** progresses through all consecutive waveforms within a given range – initial and final waveforms need to be defined,
- **random-access** progresses through a freely-defined sequence of waveforms – sequence can be altered during play.

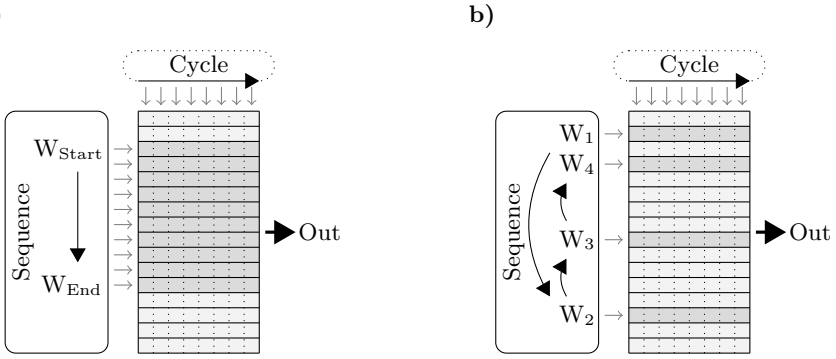


Figure 2.25. Wavetable access modes: a) sweep; b) random-access; each row represents a single wavetable, and each cell – a single signal value

More advanced synthesizers have the ability to change a sequence depending on the current envelope segment. The effect is a different source signal in attack, decay, sustain, and release phase. A simpler variant is referred to as a **loop sequence**, where each segment of an envelope is mapped to a specific wavetable, looped throughout this segment. Both techniques have the advantage of preserving duration of envelope segments while changing pitch, unlike a transposition applied in sample replay technique, which scales durations of all segments proportionally to the shift of frequency.

Interpolating Waveshapes

One wavetable can be 'morphed' into another using interpolation [485]. It allows to prevent audible discontinuities while changing timbre. It can also provide intermediate waveforms if wavetables contain only extreme examples of e.g. dynamics or other performance-related parameters [85].

Rather than crossfading between two waveforms, some form of polynomial interpolation is usually applied. One of possibilities is to use Lagrange interpolating polynomial [85] that passes through N points, from (x_1, y_1) to (x_N, y_N) [451]

$$P(x) = \sum_{j=1}^{N} P_j(x) \tag{2.24}$$

where

$$P_j(x) = y_j \prod_{\substack{k=1 \\ k \neq j}}^{N} \frac{x - x_k}{x_j - x_k} \tag{2.25}$$

The coordinates of points represent positions x_j of wavetables on some assumed scale, and sample values $y_j[n]$ in a wavetable, where n is the sample index and j is the wavetable index.

As an example we can assume that three wavetables y_1, y_2, and y_3 are equidistantly positioned (Fig. 2.26), therefore we can set $x_1 = 0$, $x_2 = 1$, and $x_3 = 2$. In this case $N = 3$ and for each signal sample n polynomial will assume the following form [85]

$$y[n] = \frac{(x-1)(x-2)}{(0-1)(0-2)} y_1[n] + \frac{(x-0)(x-2)}{(1-0)(1-2)} y_2[n] + \frac{(x-0)(x-1)}{(2-0)(2-1)} y_3[n] \qquad (2.26)$$

Controlling $x \in [0, 2]$ we can 'morph' between wavetables y_1 $(x = 0)$, y_2 $(x = 1)$, and y_3 $(x = 2)$.

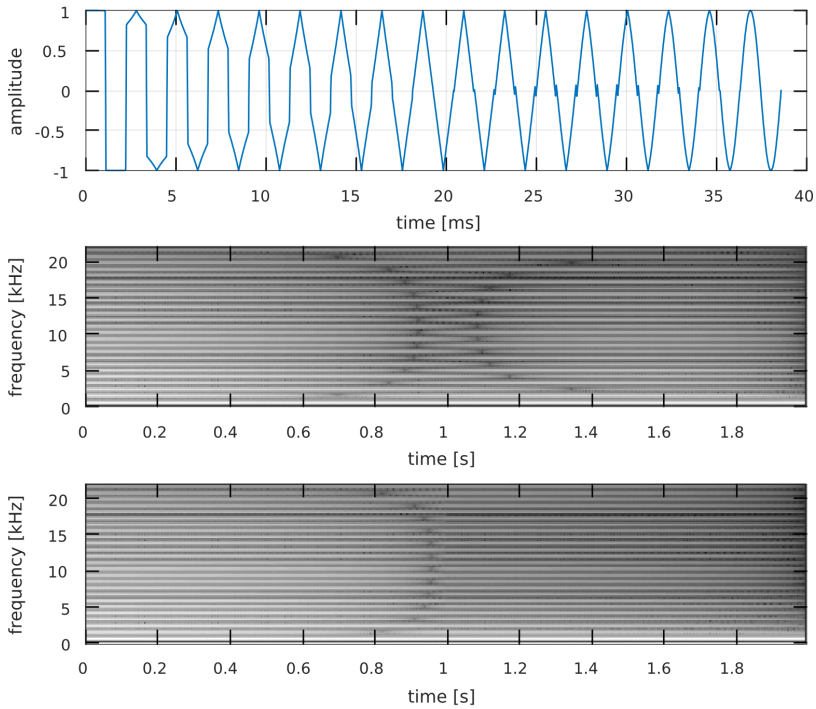


Figure 2.26. Morphing from square, through triangle, to sine wavetable using Lagrange interpolation; top plot is the output in a short time scale to emphasize waveform changes; middle plot is a spectrogram of two seconds morphing; for comparison, bottom plot shows period-based linear crossfade

Figure 2.26 presents the effect of interpolating from square, through triangle, to sine waveform, using formula (2.26). It is compared to a crossfade between wavetables. Since interpolation is carried out separately in each period, crossfade was performed on a period basis as well, i.e. weight ratio of wavetables was constant in one cycle, and changed between cycles. Although both effects are generally similar, the differences

are easily audible. In this particular case interpolation introduces high-frequency partials corresponding to a waveform region between 22 and 32 ms (top plot), where additional sharp edges appear.

2.2.1.3. Resynthesis

A general sound production principle employed by wavetable synthesis is inherited from subtractive synthesis, therefore the same resynthesis techniques might be expected to work. This, however, is the case for some wavetable implementations only, because subtractive resynthesis usually involves filter banks or otherwise highly flexible filters that many wavetable synthesizers do not have at their disposal. Part of filter functionality is transferred to the oscillator – as Bristow-Johnson points out, it is not necessary to work in spectral domain if spectral manipulation can be carried out beforehand and stored as a wavetable, which is more efficient [85].

Signal modification capabilities distributed between signal generator and filters, as well as a variety of implementations with different features, increase complexity of a process of wavetable resynthesis. Apart from subtractive-type resynthesis that does not fully exploit wavetable capabilities, some wavetable synthesizers can operate as sampling synthesizers, which makes resynthesis a default sound production technique. But again, while operating as simple sample-replay device, most wavetable facilities, such as envelopes, are not available, therefore sound control capabilities are severely limited.

Bristow-Johnson [85] proposed a set of methods for extracting wavetable data from recorded sounds, albeit categorised them as suboptimal. Nevertheless, they can be considered a basis for wavetable resynthesis. The first issue is, that wavetables have to store single periods of signal that need to be selected and isolated from the recording. At the beginning, a fundamental frequency around given time t_0 is estimated [362], e.g. using autocorrelation-based average magnitude difference function (AMDF) with window $w(t - t_0)$ wider than anticipated period and centred at t_0 [85]

$$\gamma_{t_0}(\delta) \equiv \int_{-\infty}^{\infty} \left| u\left(t + \frac{\delta}{2}\right) - u\left(t - \frac{\delta}{2}\right) \right|^2 w\left(t - t_0\right) dt \tag{2.27}$$

where $u(t)$ is the analysed signal. If Δ represents the first local maximum, i.e. $\gamma'_{t_0}(\Delta) = 0$ and $\gamma'_{t_0}(\delta) > 0$ for $0 < \delta < \Delta$, then global minimum for lags larger than Δ represents period value τ, which depends on chosen t_0 [85]

$$\gamma_{t_0}\left(\tau\left(t_0\right)\right) = \min_{\Delta < \delta} \{\gamma_{t_0}(\delta)\} \tag{2.28}$$

Value of $\tau(t_0)$ can be extracted with fraction-of-sample precision by interpolating around the global minimum.

Having determined a value of period, a single signal period around t_0 is extracted from the recording. Since when utilised as a wavetable it will be looped, it needs to be periodically extended.

Unless the signal was perfectly periodic, simple repeating extracted period will produce discontinuities that can be attenuated by applying a normalised window w_n [85]

$$\hat{u}_{t_0}(t) \equiv u(t) w_n \left(\frac{t - t_0}{\tau(t_0)} \right) \tag{2.29}$$

with complimentary fade-in and fade-out characteristic. Such window needs to satisfy the following conditions [85]

$$\begin{aligned} w_n(-\beta) &= w_n(\beta) \\ w_n(0) &= 1 \\ w_n(\beta) &= 0 \qquad \text{for} \qquad |\beta| \geq 1 \\ w_n(\beta - 1) + w_n(\beta) &= 1 \qquad \text{for} \quad 0 \leq \beta \leq 1 \end{aligned} \tag{2.30}$$

therefore e.g. Hann window can be utilised.

With time-scaled window of half-amplitude length equal to $\tau(t_0)$, periodic extension of $\hat{u}_{t_0}$ [85]

$$u_{t_0}(t) \equiv \sum_{n=-\infty}^{\infty} \hat{u}_{t_0}(t) \left(t - n\tau(t_0) \right) \tag{2.31}$$

is a match for analysed signal in t_0.

Extracted wavetables have to be phase-aligned if they are to be crossfaded. Period τ determines the fundamental frequency f_0. Assuming phase to be zero at $t = 0$, one can obtain phase at t_0 [85]

$$\phi(t_0) = 2\pi \int_0^{t_0} f_0(t) dt \tag{2.32}$$

In the end resampling [494, 312] is applied to extract arbitrary number of K signal samples from the periodic extension between $\left(t_0 - \frac{\phi(t_0)}{2\pi} \tau \right)$ and $\left(t_0 - \frac{\phi(t_0)}{2\pi} \tau + \frac{K-1}{K} \tau \right)$. The number of samples needs to be at least twice the index of the highest harmonic, but since wavetable values are to be interpolated during synthesis, it is better to further increase K [85].

The final issue is to determine how many wavetables need to be extracted, and where are they to be extracted from. The simplest approach would be to extract them in constant time intervals, and starting with small interval increase it, until reconstructed sound differs from the original. The criterion can be purely auditory, or based on a quantitative estimation, such as cumulative or maximal deviation of partials amplitudes. Some propositions can be found in works of Horner [251, 246].

2.2.1.4. Control of Pitch, Duration, and Timbre

Pitch and Duration

Both single, and multicycle table-lookup generators are based on wavetables containing single periods of signal. They do not store any evolution-related data and

thus the signal they produce is time-invariant. Through manipulation of generator frequency one directly controls pitch of produced signal, without affecting its duration.

Like in subtractive method, change of wavetable oscillator frequency shifts a whole spectrum of produced signal. Though in wavetable this effect can be more prominent due to the fact that more spectral details are produced by a generator, and less by filters, while in subtractive synthesis oscillator produces relatively simple spectrum that is later shaped by filters.

Sharing a principle of operation with subtractive method, wavetable imposes signal evolution through envelopes, thus providing means for control of signal duration. Amplitude envelope starts with a note-on event, and continues to the sustain segment which is held until note-off event arrives, triggering the last, release segment.

Timbre, Registers, Dynamics, and Articulation

Wavetable method utilises mechanisms inherited from subtractive synthesis to control pitch and duration. However, when it comes to controlling timbre, it uses a different approach. Subtractive method provides three ways to control timbre, namely choice of oscillator waveform, arrangement of unit generators, and control over extensive set of signal modifiers. Out of these, units arrangement predestines a synthesizer to a particular task, such as synthesis or resynthesis, oscillator provides a very limited choice of waveforms, and therefore signal modifiers have a predominant role. Wavetable, on the other hand, relies primarily on changing waveforms produced by a table-lookup oscillator. If an EG is employed to control interpolation of waveshapes, it produces an effect analogous to envelope control of a filter cut-off or centre frequency.

Subtractive method allows to either relate filter frequency to a fundamental frequency of produced sound, or to set it to absolute value, thus providing means of reproducing resonances of instrument corpus that do not change with pitch. In wavetable, however, independent control of pitch and spectral structure is difficult to achieve, since most of spectral shaping is performed not in filters, but in generators, and spectral structure shifts with pitch. To mitigate it, different sets of wavetables can be provided for specific instrument registers or pitch regions – the smaller the region, the better the result, but larger memory requirement and effort needed to prepare control data.

Similar solution – dedicated sets of wavetables combined with changes to envelope segments – allows to reproduce various separated articulations or timbre changes caused by changes in dynamics. Note transitions can be simulated through skipping or combining segments of envelopes. This technique may be enhanced with waveshape interpolating – thus timbre can morph from one pitch to another.

2.2.1.5. Multiple Wavetable Synthesis

Horner et al. [251] proposed a method that combines principles of wavetable and additive synthesis. It utilises a number o fixed wavetables that are mixed together. Each wavetable has its own amplitude envelope, therefore their proportions differ in time. Originally, multiple wavetable synthesis was aimed at finding close matches

to sounds of musical instruments, therefore in this context it may be considered a resynthesis method.

Figure 2.27 presents the principle of the multiple wavetable synthesis. General layout is not unlike additive synthesis (Fig. 2.9), but components are different – wavetables are used instead of sinusoids. Amplitude envelopes are also referred to as time-dependent wavetable weights. They serve a purpose of controlling wavetables proportions in output signal. It is usual, however, that in a given moment only two wavetables have non-zero weights, therefore what weights actually control is a crossfading between subsequent wavetables [470]. One, common fundamental frequency is set for all wavetables. Wavetable phases must be aligned to avoid phase cancellation while crossfading.

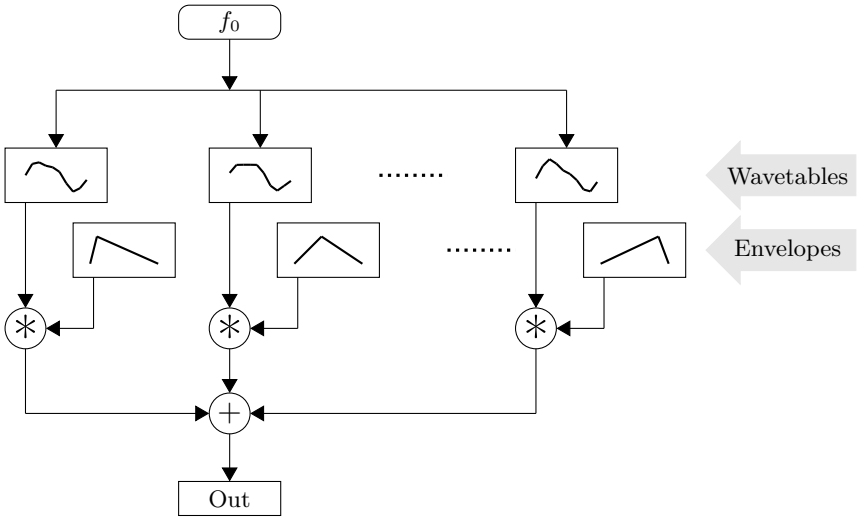


Figure 2.27. Multiple wavetable synthesis
Source: author's elaboration, based on Horner [251]

Multiple wavetable resynthesis can employ either genetic algorithms (GA) or principal component analysis (PCA) to find wavetable spectra and amplitude envelopes that mixed will match the original time-varying spectrum [251]. A task of GA was to select spectra at various points in time, while PCA helped to obtain a set of orthogonal basis spectra for wavetables. Resynthesis results were evaluated using the following relative error measure [251]

$$\bar{\epsilon}_{rel} = \frac{1}{N} \sum_{j=1}^{N} \left(\frac{\sum_{k=1}^{M} \left(A_k(t_j) - A'_k(t_j) \right)^2}{\sum_{k=1}^{M} A_k^2(t_j)} \right)^{\frac{1}{2}} \tag{2.33}$$

where N is the number of selected time values, t_j are time values, M is the number of harmonics, $A_k(t)$ is the amplitude of original signal k-th harmonic, and $A'_k(t)$ is its equivalent in synthesized signal, according to [251]

$$A'_k(t) = \sum_{l=1}^{P} w_l(t) a_{kl} \tag{2.34}$$

where a_{kl} is the time-fixed amplitude of k-th harmonic of l-th wavetable, $w_l(t)$ is the weight of l-th wavetable, and P is the number of wavetables. In case of both, GA and PCA, three to five wavetables proved enough for a good reproduction of the original signal, however with lower number of wavetables GA produced smaller errors.

A principle similar to multiple wavetable synthesis has been implemented in a number of synthesizers. These implementations differ in the arrangement of wavetables as well as their control scheme, and they use various names of the technique, such as **wavetable crossfading**, **compound synthesis**, **vector synthesis**, or **linear arithmetic synthesis** [470]. Wavetable crossfading variant uses several wavetables to crossfade from the first to the second, from the second to the third, and so on – so that pairs of adjacent wavetables are mixed, one fading-in, the other fading-out, thus producing rich, evolving sound. Crossfading control can be automatic or manual. Automatic uses envelopes, while in manual weights can be controlled with a joystick.

Wavestacking synthesis may be seen as a hybrid of additive, multiple wavetable, and sampling synthesis. While it retains the general layout of multiple wavetable, it does not use looped periods of signal, but instead utilises full sound samples that are enveloped and mixed. Unlike in multiple wavetable, where common fundamental frequency is assumed, component signals in wavestacking can have different frequencies, or – more precisely – each can have its own frequency envelope as a characteristic of recorded signal, and can be additionally resampled. Wavestacking may be combined with multiple wavetable synthesis.

2.2.1.6. Wave Terrain Synthesis

In digital implementations waveform is ordinarily stored in a one-dimensional data structure, indexed by a time-related variable. Wavetable synthesis uses this approach to store periods of signal as one-dimensional arrays. However, if the data arrangement had changed, this alone might serve as a basis of a sound synthesis method.

A method referred to as wave terrain synthesis expands wavetable to two dimensions, which results in new capabilities, problems, and possible applications. In a two-dimensional domain waveform becomes a wave terrain, and a path chosen over wave terrain to produce one-dimensional audio signal is referred to as an orbit or trajectory. According to Roads [470], the term 'wave terrain' was first used by Gold [64], and a number of implementations followed [378, 76, 265].

Terrain

A wave terrain can be considered a surface over a two-dimensional domain – a sample value within a wave terrain is addressed by two indices. Terrain data can be generated through some mathematical formula, or can originate from an arbitrary

source such as wavetable cross-multiplication, image, relief map, texture, etc. Mathematical formulas have the advantage of predictable output signal characteristics.

If a continuous output signal is to be produced, and the orbit is expected to cross terrain boundaries, two conditions should be met. Firstly, terrain generator functions in both directions and their first-order partial derivatives should be continuous. Secondly, generator functions should be zero, or at least assume a constant value, on edges of terrain – it will allow to cycle wave terrain in both directions [470]. Works of Mitsuhashi [378], Borgonovo [76], and James [265] provide definitions of functions fulfilling both conditions in the range $x_1 \in [-1, 1]$, $x_2 \in [-1, 1]$. A common example is the following function, presented in Figure 2.28

$$y(x_1, x_2) = (x_1 - x_2)(x_1 - 1)(x_1 + 1)(x_2 - 1)(x_2 + 1) \tag{2.35}$$

where x_1 and x_2 are coordinates.

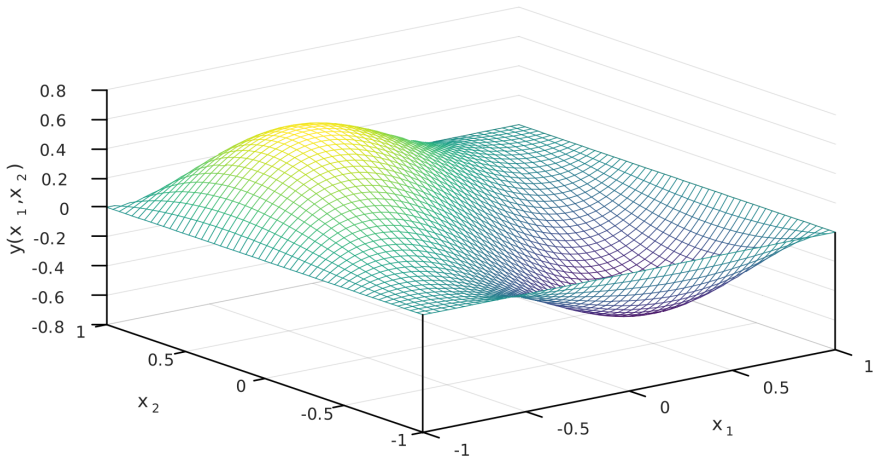


Figure 2.28. Wave terrain generated by (2.35) [265]

However, complying to the aforementioned restrictions it is difficult to produce more complex surfaces. James [265] gathered a large number of procedures that allow to generate useful terrains either by violating the conditions, or by going beyond simple arithmetic formulae. He pointed at particular characteristics that produce interesting and desired auditory effects, such as discontinuities that introduce large number of harmonics, or undefined values. For instance, the following function [265]

$$y(x_1, x_2) = \cos\left(12 \sin\sqrt{(x_1 - 1)^2 + x_2^2} - 4 \tan^{-1}\left(\frac{x_2 + 1}{x_1}\right)\right) \tag{2.36}$$

is periodic, but discontinuous (Fig. 2.29). In a digital domain the discontinuity itself can be simply avoided, while its steep surroundings remain useful still.

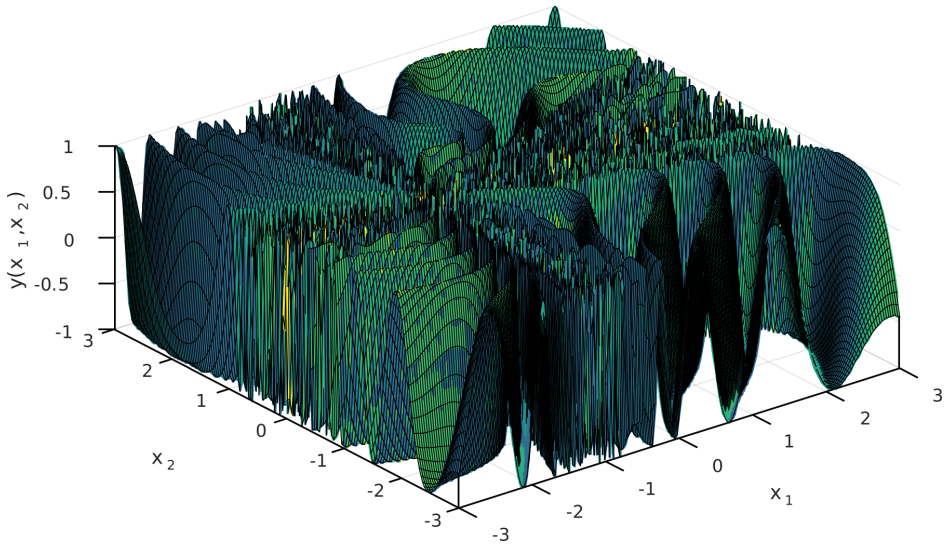


Figure 2.29. Wave terrain generated by (2.36) [265]

Orbit

In a process of wave terrain synthesis a two-dimensional data structure has to be rearranged into a one-dimensional output signal. This task is performed by an orbit. It is an orbit that determines whether an output signal will be periodic, or not – signal periodicity follows a periodicity of an orbit. Therefore, an orbit controls a fundamental frequency. Orbits can be divided into four categories presented in Table 2.3.

Table 2.3. Categories of orbits in wave terrain synthesis, according to James [265]

Category	Examples
Periodic	Lines, ellipses, Lissajous curves
Quasi-periodic	Spirals, waveforms of recorded musical instruments
Chaotic	Strange attractors, waveforms of recorded environment sounds
Stochastic	Noise, random walks

A rich collection of orbits can be found in the work of James [265]. A rose curve (Fig. 2.30, top left)

$$r = a\cos(n\theta) \tag{2.37}$$

controlled by parameter n is an example of periodic orbit. A butterfly curve (Fig. 2.30, top right) is another example from the same category

$$r = e^{\cos\theta} - 2\cos(4\theta) + \sin^2(\frac{\theta}{12}) \tag{2.38}$$

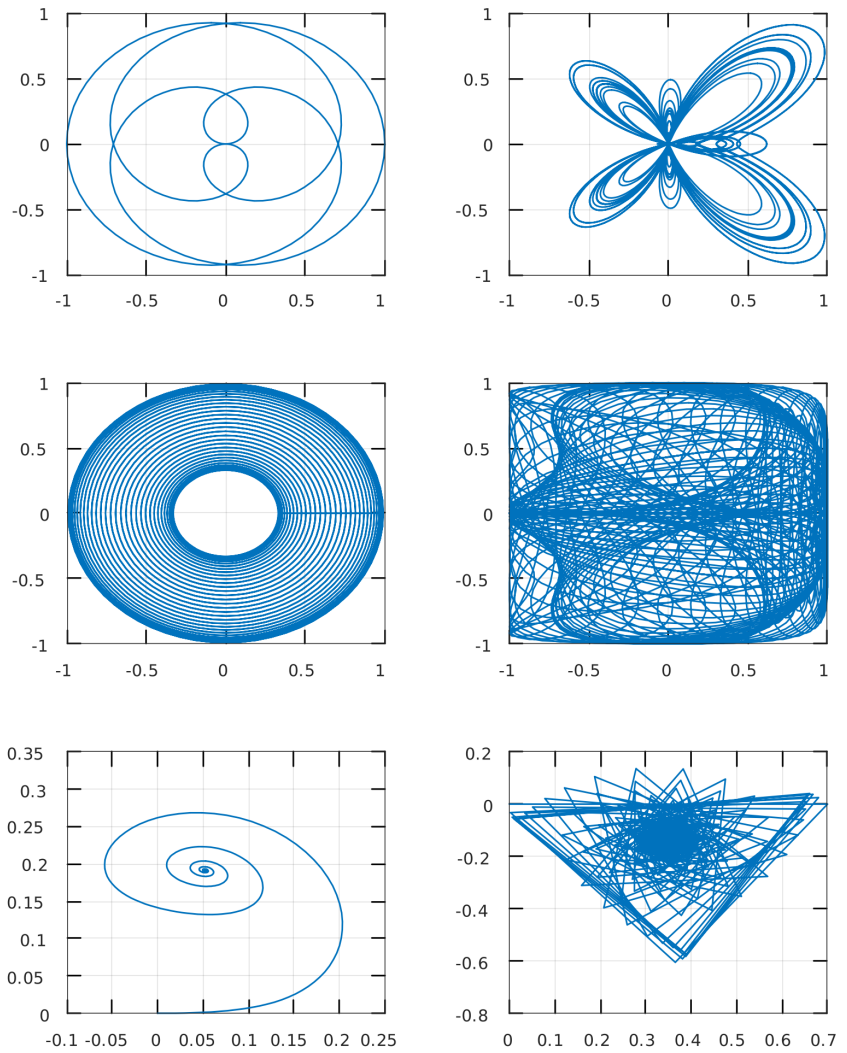


Figure 2.30. Orbits for wave terrain synthesis [265], from top-left to bottom-right: rose-curve (2.37); butterfly (2.38); standard torus (2.39) projected on x_1x_2-plane; figure 8 torus (2.40) projected on x_1x_3-plane; and Ikeda map (2.41) with two different sets of parameters

A standard torus

$$\begin{aligned} x_1 &= (a + b\cos\theta)\cos\phi \\ x_2 &= (a + b\cos\theta)\sin\phi \\ x_3 &= c\sin\theta \end{aligned} \tag{2.39}$$

projected on a plane can produce a quasi-periodic orbit (Fig. 2.30, middle left). Similarly, a figure 8 torus can be utilised (Fig. 2.30, middle right)

$$\begin{aligned} x_1 &= \cos\theta(a + \sin\phi\cos\theta - \frac{1}{2}\sin\theta\sin(2\phi)) \\ x_2 &= \sin\theta(a + \sin\phi\cos\theta - \frac{1}{2}\sin\theta\sin(2\phi)) \\ x_3 &= \sin\theta\sin\phi + \frac{1}{2}\cos\theta\sin(2\phi) \end{aligned} \tag{2.40}$$

where a, b, and c are parameters.

Various chaotic orbits can be generated using the Ikeda map (Fig. 2.30, bottom row)

$$\begin{aligned} x_1[n+1] =& a + b(x_1[n]\cos(x_1[n]^2 + x_2[n]^2 + \phi) - \\ & - x_2[n]\sin(x_1[n]^2 + x_2[n]^2 + \phi)) \\ x_2[n+1] =& b(x_1[n]\cos(x_1[n]^2 + x_2[n]^2 + \phi) + \\ & + x_2[n]\sin(x_1[n]^2 + x_2[n]^2 + \phi)) \end{aligned} \tag{2.41}$$

Evolution

Most common periodic orbits are elliptical. If a signal is supposed to evolve in time, then one of non-periodic orbits needs to be applied. An interesting case of orbits are projections of higher-dimensional objects onto a plane. It is also possible for an orbit trajectory not to be pre-calculated, but entered manually by a user, e.g. with a help of graphic tablet, touchscreen, or similar device. A temporal evolution of orbits can be achieved through geometric transformations, such as scale, translation, rotation, or reflection [265]. Orbits may also be arranged in poly-trajectories, where two or more of them are added or multiplied – one is usually responsible for periodicity, thus changes fast, while the other is slowly modulating position of the first one.

A straightforward way of determining spectral evolution is possible only for relatively simple wave terrains and orbits. It is usually recommended that if one, terrain or orbit, has a complex structure, the other should be simple, otherwise control of output signal is practically impossible. Apart from complexity, any functional relationship between terrain, orbit, and output signal data is obviously possible only if the first two are expressed in functional form. In other cases predictability is problematic, and only general remarks can be provided [265].

Early works tended towards fixed wave terrains and variability based on orbits, but faster computers with large number of controllers allow to break this scheme [369, 367, 366, 128]. Controllers such as MATRIX, designed by Overholt [418, 419], are an efficient way of changing a wave terrain in a flexible way – in case of MATRIX through 144 independent continuous controllers arranged into 12×12 array. Terrain evolution can also be imposed in a parametric manner, through a much narrower set of variables, with a use of more traditional controllers. Evolution of terrain differs from evolution of orbit in regards to rate. While orbits are processed with audio rate, wave terrains require only haptic rate, or control rate. Newer works propose a much

more complex interpretation of wave terrain data, aimed at spectral spatialisation in acousmatic music. Studying spatiomorphology and spectromorphology [515], James proposes mapping strategies within wave terrain synthesis framework that would produce meaningful – i.e. taking into account psychoacoustics – spectral and timbre spatialisation [267, 266].

2.2.1.7. Progress of Wavetable

Wavetable synthesis was implemented as a fully-developed method in hybrid synthesizers like PPG Wave or Waldorf Microwave, and progressed in a number of directions (Fig. 2.31). Its basic form is often combined with a manual wave-data entry, which resembles early wavecycle oscillators equipped with switches. In place of switches users are able to draw, or otherwise create and edit wavetable data, which results in a very direct control of sound – partially due to a very limited signal processing outside of a generator. Usually it is a part of some larger music performance systems, like in case of the *Laptop Orchestra* [516].

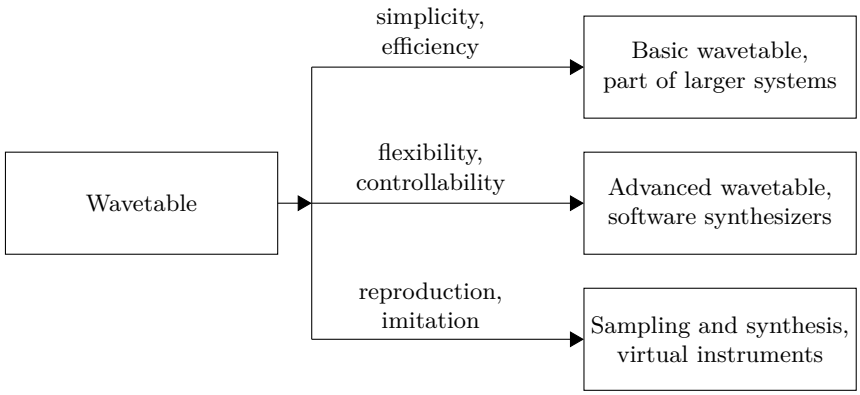


Figure 2.31. Specialisation of wavetable synthesis

The second way is blending with other methods in commercial software synthesizers. Due to similar principle, and possibility to reuse the same signal modifiers like LFOs, filters, and EGs, arranged using modulation matrix, wavetable if often mixed with subtractive synthesis. Commercial implementations employ all of multiple wavetable techniques. Single-cycle generators are supplemented with elements of vector synthesis, i.e. user controls crossfading between a small number of wavetables, usually four, with some equivalent of joystick. Wavestacking of e.g. 16 wavetables is the second option. In another one, different wavetables can be assigned to specified parts of amplitude envelope, with signal crossfading or interpolating through successive tables to produce evolving spectra – in software synthesizers this technique is often referred to as wavetable morphing[6]. Apart from multiple wavetable techniques,

[6]Simpler synthesizers allow to use only 16 wavetables, but more advanced raise this number to 256.

virtual analog filters and effects are employed to emulate sound and behaviour of hybrid systems like PPG or Waldorf instruments, where generators were implemented digitally, but filters were analogue. Some commercial synthesizers allow to load user-provided wavetables, but many are equipped with a set of built-in waveforms only. More advanced synthesizers allow to break imported audio files into wavetables, generate and edit wavetable data using mathematical formulae, or produce waveforms using additive synthesis. They also use more refined resampling algorithms. An interesting function is a real-time wavetable manipulation. It applies various modulation techniques to a table data. In a few cases wavetable is supplemented with granular synthesis.

The last way leads from wavetable synthesis to sampling. Multi-cycle oscillators with a very large number of wavetables reproduced in a fixed sequence produce an auditory effect very close to sampling synthesizers. Early wavetable synthesizers had a very limited memory, but soon it ceased to be a limiting factor, and practically whole sound events could have been stored in a series of wavetables. In transitional synthesizers attack phase could have been reproduced as a simple sound sample, while only further phases, particularly sustain, used a wavetable method [61]. Gradually, this way led to development of sampling synthesizers with much more powerful signal modification capabilities than simple sample-replay devices, i.e. with envelopes, filters, and modulators. In a number of works this method is referred to as 'sampling and synthesis' [485], It is clearly distinct from wavetable method. Its data storage is not organised in period-based tables, which has consequences for pitch, duration, and timbre control capabilities.

2.2.2. Sampling

The term 'sampling' refers to various techniques and applications often utilised in the area of contemporary sound synthesis. Applied in the role of a synthesis method directly, it may be referred to as 'sampling synthesis'. However, it raises a certain degree of controversy, and not all may consider sampling a 'true' sound synthesis technique [485]. Such claim may have a merit in case of the simplest applications of sampling that operate by directly replaying recorded sound events. In musical applications, however, such simple approach is very rare, since it provides virtually no control over reproduced sound other than start time, hence can be used only for some basic percussive purposes.

The broadest definition of sampling would be a manipulation and reproduction of recorded sounds [470]. Such experiments within musical performances have been carried out already in 1920s by Paul Hindemith, Darius Milhaud, and Ernst Toch [185]. The underlying technology was based on phonographs with controllable speed, which allowed manipulations of pitch. After the Second World War Pierre Schaeffer and Pierre Henry turned to tape recorders. A change of data carrier made it possible to edit sound in a time domain, which involved cutting, splicing, and rearranging tape fragments. Thus the *musique concrète* – a technique of working with sound objects – has been invented.

While sampling synthesis is a descendant of *musique concrète* [521], it is also related to instruments, developed since 1930s [470], that reproduced waveforms of pre-recorded real-world sounds written onto optical discs or tapes. These instruments, including Welte's light-tone organ, Sammis's singing keyboard, and much later Chamberlin, Mellotron, Optigan, Orchestron, and Birotron, to name a few, may be considered analogue sampling synthesizers.

These early sample-based instruments utilise a number of techniques important for all of later sampling synthesizers. Firstly, some of them – disc based, and a few tape-based – play samples in a loop, thus allowing to fully control note duration, and not only to shorten it. However, in analogue instruments looping is introduced at the expense of losing natural attack phase of the original instrument. Secondly, most of them use separately recorded waveforms for every pitch [485], therefore they reproduce natural register-based timbre changes. This may be considered an equivalent of modern digital sampling synthesizer full multisampling technique.

With transition from analogue to digital technology, some of more advanced capabilities – such as separate sample for every pitch – were initially lost, mainly due to very limited memory capacity in early digital devices. However, a new feature has been added: early sampling-based instruments introduced a built-in recording capability. Thus a user was able to create own samples using microphone or line recording.

Interestingly, recording capability marked a division point in sampling-based synthesis methods. **Samplers** started to rely on a record-store-replay principle, with control over sound characteristics limited to selection of a recording. **Sampling synthesizers** did not allow to record own samples, but provided libraries of recordings to use, and gradually introduced some of signal modification techniques based on other synthesis methods to grant user some degree of control over produced sound, other than basic selection from a limited set of samples. The method applied by samplers is sometimes referred to as '**sample replay**', or 'sample and replay', while sampling synthesizers are deemed to utilise '**sample and synthesis**' (S&S) method [485].

While many studies seem to acknowledge this differentiation, it recently seems to be gradually fading out. The change is related to transfer of a synthesis process from external devices and separate instruments to a very particular software form. Majority of contemporary synthesizers, not only sampling-based, are implemented as plug-in modules for digital audio workstations (DAWs). DAWs, in turn, are mainly focused on sound recording, which makes samplers built-in recording capabilities obsolete. It is sufficient that sampling synthesizer can import recordings in some standard format, and the whole system consisting of DAW and sampling synthesizer plug-in operates as an advanced mixture of sampling synthesizer and sampler (Fig. 2.32).

In a digital domain two synthesis methods, i.e. sampling and wavetable, tend to be confused with each other. Both are based on a similar principle of reproducing memory content – an array of signal samples. There is, however, a significant difference. Wavetable synthesis stores either one, or a number of signal periods, but there is always a simple relation, that a single table stores one period of a signal. In consequence, wavetable acts as a signal generator that can be paired with EGs, thus allowing to control pitch and duration separately. Wavetable can store and reproduce a natural spectrum of an instrument, but not its evolution, which can only be ap-

proximated using envelopes and interpolation. Sampling synthesis, in turn, stores the entire sound event, with its natural evolution. Timbre, duration, and pitch, however, are linked, and it requires complex tools to control them separately. In particular, depending on implementation details, control over timbre and its evolution is either completely impossible, or rudimentary at most. Notwithstanding, the difference between wavetable and sampling is clearly apparent only in case of synthesis methods principles. The actual synthesizers, apart from early models, often mix properties of both. For instance, they use samples for definite transients, such as an attack, or – in case of some instruments – release phase, and wavetables for a sustain section. It is possible, because underlying implementation technique is very similar in both cases.

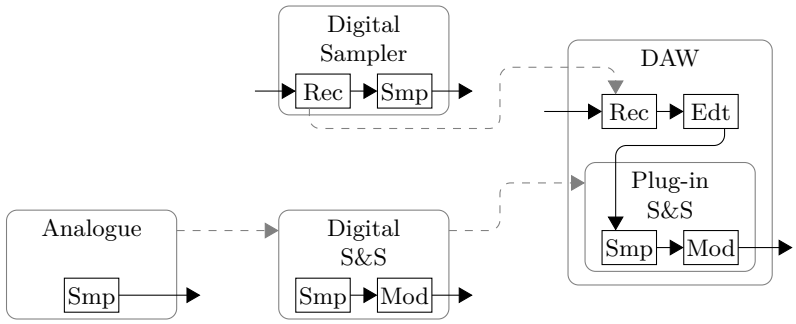


Figure 2.32. Diversification and convergence of sampling-based synthesizers; dashed gray lines indicate transfer of functions to newer technology; functions: Smp – sample storge, Rec – recording, Mod – sample modifying, Edt – sample editing

Synthesizers often supplement sampling synthesis with components of other methods, making a strict attribution of particular synthesis aspects to sampling method a subject of discussion. Despite that fact, in contemporary musical applications sampling synthesis has a very distinct and simple meaning. It is a method that on demand replays recording of a sound event representing performance of a single note, with a single pitch. Obviously, it can reproduce unpitched sounds as well, like some percussive or effect sounds, but still, they are closed sound events, equivalent to single, separated notes. Recordings contain a natural evolution of all sound characteristics, i.e. timbre, amplitude, and pitch, although the latter is usually expected to vary only slightly, like in *vibrato*. The evolution, however, is almost impossible to be removed, separated, or modified without resorting to fairly advanced signal processing. Thus in a basic sampling method a sound sample can be triggered by a note-on event, and possibly terminated by a note-off, but in between it generally does not respond to control.

Paradoxically, the greatest strength of the sampling method is also its weakness. Sampling may effortlessly produce sounds characterised by even the most complex evolution of pitch, timbre, and amplitude, and it requires no parameters to do so – the evolution is simply reproduced. However, it cannot be controlled otherwise, than by switching to another sample. In effect, the set of controllable parameters in basic sampling is significantly smaller than in other synthesis methods.

It is limited to:

- pitch,
- duration,
- amplitude,
- and selection of sample.

2.2.2.1. Digital Sampling Synthesis Principle

Figure 2.33 presents an overall diagram of digital sampling synthesis. It can be broken down into two parts: sample preparation (Tab. 2.4), and sample playback. Firstly, a source sound is chosen and sampled, i.e. converted to a digital recording. Secondly, it is edited, e.g. to remove leading and trailing silence. If a sample is to be looped, loop points need to be defined, either manually or automatically. When sampling and editing is carried out in a DAW, sample can be processed as any normal recording, using all of extensive DAW capabilities. Alternatively, a sample may not be recorded at all, but instead produced using some other synthesis plug-in. Depending on data format handled by sampling synthesizer, some additional information may need to be provided, e.g. regarding special purpose sample zones. This concludes the first part – sample is stored in memory, and ready to use.

Table 2.4. Basic operations carried out to prepare a sound sample

Operation	Description
Acquisition	A sample is either recorded, read from file, produced using a sound synthesizer, or created using other signal generator
Setting pitch	Assigning pitch to a sample; a sampler requires this information to perform pitch-shift when a sample is to be replayed at pitch different than assigned
Trimming	Leading and trailing silence is removed; to avoid clipping on both sample ends, removed sections should be aligned to zero crossings
Applying fades	If signal gradually fades into trailing background noise, a fade-out may be applied; similar operation may be required in case of a gentle attack emerging from background noise
Normalisation	Signal level is normalised to match other samples within a set
Setting loop points	Loop points allow a section of sample between them to be repeated in order to control playback duration

The second part starts with a sample in memory. It is read on demand, e.g. in case of a note-on event. If a pitch assigned to a sample differs from a target pitch, resampling is performed using table-lookup technique (2.23). Before playing it back, a sample may be further processed using signal modifiers from other synthesis methods, such as envelopes, filters, various modulators, and effects.

Figure 2.33 depicts data paths for typical arrangements of sample replay variant (in red) and S&S (blue). The former consists of both parts, sample preparation and playback, though it skips signal modifying stage. Thus its only means of controlling

timbre is to select or record another sound. The latter starts with ready to use samples, and allows to control some of their characteristics, thus making up for the lack of recording capabilities.

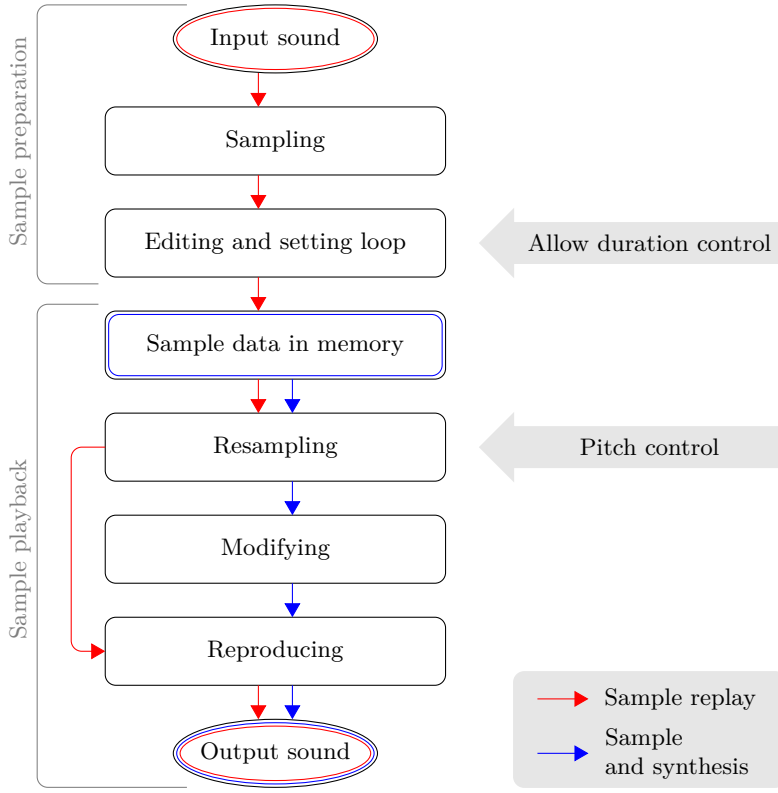


Figure 2.33. Stages of digital sampling synthesis; red path denotes sample replay variant, blue – sample and synthesis

Presented layout is characteristic for a basic implementation of sampling method, and may differ in more advanced cases. Improvements addressing deficiencies of basic sampling method are concentrated in two areas: control capabilities, and imitation quality. The former deficiency is obvious. Risset attributes it to roots of sampling reaching *musique concrète*, which resorted more to a technique of *collage*, than to actual control and manipulation of sound itself [464]. The latter deficiency, however, may seem questionable, since sampling reproduces recorded sounds, and digital recordings can be of exquisite quality. The problem is not in recordings though, but in their content and its handling. Separately recorded pitches do not reproduce fluent musical phrases well enough, since pitch transitions are not preserved. Therefore, while samples may sound perfectly natural played separately, they lose this quality when combined into a phrase.

2.2.2.2. Control of Pitch

In additive, subtractive, wavtable, and many other sound synthesis methods, fundamental frequency is either a direct parameter, or it can be trivially calculated. For instance, in wavetable synthesis it is defined by table length, phase index increment, and sampling frequency (2.23). Despite apparent similarities between wavetable and sampling, pitch control in sampling is less straightforward and involves additional effort.

A sound sample does not store one period of signal that would allow to precisely determine its fundamental frequency. It may have, however, some internal pitch. Without knowledge regarding contents of a sample this pitch is unknown. It needs to be determined and attributed to a sample during its preparation. Moreover, it may naturally, albeit slightly, change over time, e.g. due to *vibrato*, or during attack phase. Therefore, there is not one fundamental frequency, but a function $f_0(t)$, even though variability is usually much below a semitone range, so that a sample may be attributed with one averaged pitch. This variability is desired and needs to be preserved.

Sampling synthesis may use two general means to control pitch of produced sounds: **pitch-shifting** or **multisampling**. Pitch-shifting allows to produce various pitches out of a single sample. Multisampling uses separately recorded samples to reproduce different pitches – depending on target pitch, matching sample is reproduced. Pitch-shifting allows to control pitch in a continuous manner, but multisampling is only able to provide discrete values, since every pitch requires a separate sound sample. In most cases samples can be recorded in semitone intervals at best[7]. Therefore even if multisampling is the base pitch control mechanism, fine tuning, if required, has to resort to pitch-shifting anyway.

Pitch-shifting

Pitch-shifting can be carried out using various methods. The simplest way is to change a digital to analogue converter (DAC) clock frequency – it does not require any changes applied to the sample. Manipulation of DAC clock, however, is a viable option only if each synthesizer voice is reproduced through separate DAC. Such design was only encountered in early digital samplers, with analogue mixers. With a common DAC, all voices would be shifted together. Contemporary implementations, and particularly software synthesizers, utilise a common DAC and digital mixing, thus a different method is required.

The most common pitch-shifting method is a conversion of sample rate, or resampling. As a result of conversion, the same waveform is represented by a different number of signal samples. Therefore, if sample rate remains unchanged, frequency of a signal will be scaled by a ratio of original and new number of signal samples

$$f_r = f_o \frac{N_o}{N_r} \tag{2.42}$$

[7]Most instruments that utilise keyboards or frets can only play pitches in semitone intervals, unless they are detuned. It would be arduous and impractical to attempt at recording pitches between semitones.

where f_r is the signal frequency after resampling, f_o is the original frequency, N_r is the number of signal samples after resampling, and N_o is the original number of signal samples within a sound sample.

A pitch may need to be shifted by an arbitrary interval, including microintervals[8], or by even smaller values, in case a pitch-bend controller is to be utilised to produce *portamento* effect. Due to this reason a common approach is to calculate new samples using interpolation, though it introduces some level of distortion. Software synthesizers may allow to choose order or type of interpolation, thus a compromise between signal quality and processing requirements may be found.

A comparison of the effect of changing DAC clock frequency and resampling is presented in Figure 2.34. The former keeps the number of samples, and the latter – the sample rate. Using either method, a sample is shifted with its internal pitch envelope, and the operation affects all the component frequencies. They are shifted by the same frequency ratio, while their mutual amplitude ratios remain unchanged. In effect both methods have a side effect of shifting not only a pitch, but also an entire spectral structure and its evolution, which may not be desired. Samples of acoustic instruments shifted by more than a few semitones sound unnatural due to retained spectral envelope of a different register. Perhaps even more important consequence of pitch-shifting using either method is a change of sample duration, proportional to a frequency shift. Consequently, internal time-structures, such as the attack phase, or *vibrato*, are unnaturally slowed down or sped up.

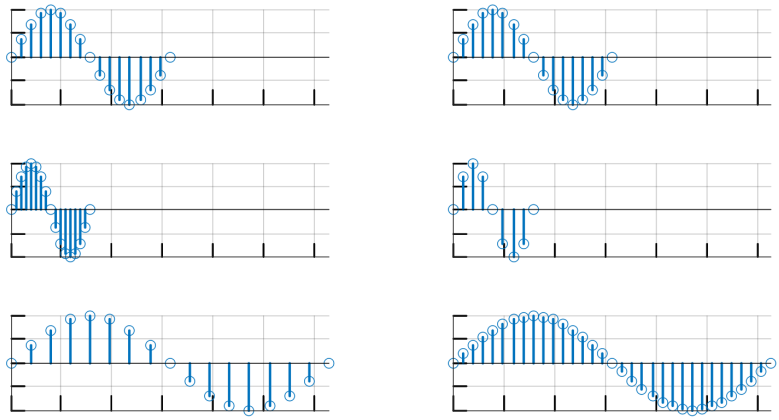


Figure 2.34. Comparison of pitch-shifting through change of DAC clock frequency (left), and through resampling (right); top – original pitch, middle – shifted octave up, bottom row – shifted octave down

Some of resampling-related problems may be solved by using more elaborate pitch-shifting methods. Roads [470] discusses various means, such as granular time-domain

[8]Microintervals are intervals smaller than a semitone.

techniques, wavelets, and two resynthesis techniques: phase vocoder and linear predictive coding.

Digital **time-granulation** allows to shift pitch without change of duration. The technique is based upon a principle of an electromechanical device built by Denis Gabor. The device used a head spinning across the film or tape recording, which produced signal grains, i.e. smoothly windowed fragments of the original signal. Grains were reassembled on another tape. By controlling tape speed and spinning of head, pitch and duration could have been adjusted separately. Head rotation velocity controlled signal duration without affecting its pitch. Change of pitch without change of duration was achieved through altering a tape speed, and compensating for the resulting duration change with appropriate change of head rotation velocity. The device produced audible discontinuities between grains if signal values in junction points were not matched. Several digital variants of the procedure were proposed. Jones and Parks [273] applied smooth grain envelopes with a small overlapping area, and achieved a continuous reconstruction. As in Gabor's device, local frequency in grains is preserved, while duration is changed by multiplying or removing some of grains.

Similar result, i.e. pitch-shift without affecting duration, may be achieved through application of phase vocoder. Evolution of signal partials is determined by calculating FFTs of overlapping signal segments. Frequencies of partials are multiplied according to required pitch shift, and such modified data is used to control additive synthesis. The procedure does not affect signal duration. Due to intermediate stage with data in spectral form, it is possible to partially correct spectral structure, according to knowledge regarding registers or other spectral characteristics of shifted instrument [168]. Instead of FFT analysis, it is possible to use wavelet transform [310], and scale phase values of analysed wavelets.

Linear predictive coding [388, 167, 318] may also be used as a pitch-shifting technique. However, due to its filtering-based resynthesis mechanism, it does not produce perfect reproduction. On the other hand, its frame-attached data that e.g. distinguishes between voiced and unvoiced segments of signal, may allow to correct or modify more aspects than other methods.

Sample rate may also be converted using **fractional delay (FD) filters** discussed by Välimäki [569]. The method, presented by Tarczynski [547], was further discussed by Franck [193] and Blok [70, 71]. Algorithm proposed by Blok allows to change resampling ratio freely and instantaneously, during processing, thus providing means for continuously changing pitch-shift. If $r[n]$ is the inverse of instantaneous resample ratio

$$r[n] = \frac{f_{s1}[n]}{f_{s2}[n]} \tag{2.43}$$

where n is the sample index in output signal, $f_{s1}[n]$ and $f_{s2}[n]$ are input and output sample ratios, respectively, then the fractional delay is a distance from the output sample to the nearest input sample, calculated using a recursive formula [71]

$$d[n] = d[n-1] - r[n] + \Delta m[n] \tag{2.44}$$

where $\Delta m[n]$ is the number of new input samples required to calculate output sample [71]

$$\Delta m[n] = \lfloor r[n] - d[n-1] \rceil \tag{2.45}$$

where $\lfloor \cdot \rceil$ is the nearest integer, or round function. The algorithm starts with $d[0] = 0$, $\Delta m[0] = 0$, and the buffer filled with zeros. Then, it waits for $\Delta m[n]$ new input samples. Next, output sample $u[n]$ is calculated using FD filter with fractional delay $d[n]$. In the last step, $d[n]$ and $\Delta m[n]$ for next n are calculated, and the algorithm goes back to wait for next $\Delta m[n]$ new input samples. The algorithm needs to calculate different values of $d[n]$ for every output sample. If resampling ratio is rational, the process is periodic, and values can be stored in a look-up-table [234]. In case of arbitrary, time-dependent ratio, filters need to be calculated in run time, which may be acheved by using the Farrow structure [188, 232]. Blok proposes to design FD filters using the offset window method [613, 614], since it reduces large lobes in a stop-band in comparison to minimax filters [312].

Pitch Multisampling

Control over a sound produced by sampling synthesis is very limited. A possible solutions to this deficiency is to relegate control outside of a synthesizer, and record larger number of sound samples with desired qualities for a given instrument. With such set of samples available, a control over particular sound quality is carried out through selecting appropriate sample. Such approach is referred to as multisampling [253], and can be applied to virtually any sound characteristic. Clearly, it has two important disadvantages:

- control is not continuous, and only as fine, as large a sample set is,
- it can be issued only when a sound is to be started, with no possibility to change it while a sample is played.

One area where multisampling can be advantageous, is a pitch control. Apart from pitch-bending or ribbon controllers, main means to select a pitch in synthesizers is a piano-like keyboard, where only discrete pitches, distributed in semitone intervals, are represented. Actually multisampling had been already applied to control pitch in analogue sampling synthesizers. Mellotron had individual mechanisms attached to all keys. Each mechanism utilised a strip of magnetic tape with a recording of different pitch, reproduced by appropriate key. Digital samplers can utilise the same principle.

Multisampling applied to control pitch (Fig. 2.35) solves an issue of register-related change of timbre, characteristic for simple pitch-shifting methods, and only partially attenuated by applying more advanced methods based on analysis-resynthesis. Moreover, if a selected few of instrument pitches have some characteristic features, not shared by their neighbours[9], a full multisampling will preserve them.

Number of available pitches depends on playing range of a particular instrument, and in same cases can be rather large. For instance, a grand piano has at least 88 keys. It has two consequences. Firstly, multisampling may require a large amount of sample

[9] It can be e.g. a characteristic inharmonicity, or buzzing sound produced by some excited element.

memory. Secondly, in case of acoustic instruments with human performer, recording and preparing large sample sets may be a laborious and difficult enterprise. As for memory requirements, it is not an issue in contemporary software synthesizers – at least not for pitch multisampling alone. Assuming that recordings are monophonic, sampled with 44.1 kHz rate, stored in 24-bit values, and that an average sound sample lasts 10 seconds, one terabyte drive can hold over 750 thousands of them. However, preparing such large sample libraries is still a problem. During recording all pitches have to be performed very consistently. Variations in playing technique, i.e. dynamics or articulation, affect the timbre, which cannot be well compensated later, during editing [253]. Similarly, editing large sample sets takes a lot of effort, since every sample has to be at least normalised, trimmed, and searched for adequate loop points.

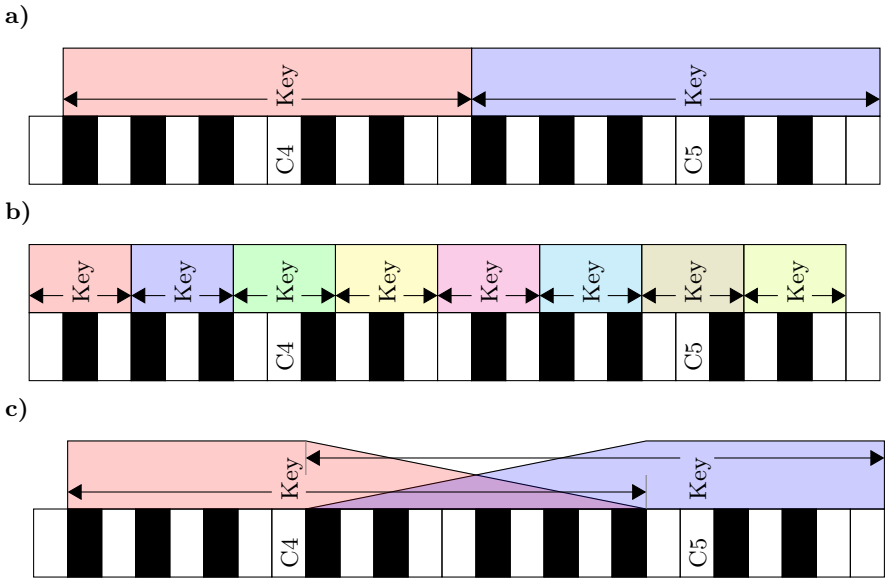


Figure 2.35. Examples of key-maps: a) octave-wide key-zones (represented by different colours) with C key-notes; b) minor third zones; c) overlapping zones

Control over pitch through multisampling is limited to sampled values only. If fine tuning is required, e.g. to perform a *portamento*, synthesizer has to resort to pitch-shifting. Taking this a step further, a synthesizer can use a lesser number of samples, separated with intervals larger then semitones, and produce missing pitches by pitch-shifting, or transposition. Organisation of samples and their transpositions is provided by a mechanism of key-maps. A map attributes samples with recorded pitches to particular keys. These are the key-notes. Remaining keys are divided into key-zones around key-notes. Zones are also referred to as ranges or groups. If a key-note is used, it is played without pitch-shifting. If a key from a zone is pressed, its key-note sample is appropriately transposed, and reproduced. Such technique is also referred to as multisampling, while a variant with all pitches sampled is sometimes

distinguished as 'full multisampling'. Using key-notes and zones is a common way to preserve some of full multisampling qualities, but with much less effort put into preparation of samples.

Key-notes do not have to be evenly distributed, and differently sized key-zones may be utilised in one key-map, since not all registers of sampled instrument have to be equally important. In simple scenarios it is common to use one or two key-notes per octave. However, a satisfactory effect, with no easily audible formant shifts, can be achieved with four samples per octave, distributed every minor third interval. In such arrangement the largest transposition required is by semitone (Fig. 2.35). Samplers may allow to create overlapping zones to attenuate abrupt changes of timbre between samples. In overlap section two neighbouring samples are mixed according to defined ramps. However, mixing signals with the same fundamental frequency may lead to phase cancellations.

2.2.2.3. Control of Timbre

Multisampling can be utilised to introduce some degree of control over timbre. In most acoustic instruments change in dynamics implies a change in performance technique. This, in turn, causes a change of timbre. A common scenario is to sample several levels of dynamics, and map such obtained samples to key velocity values, so that lower velocity reproduces a sample recorded in *piano*, and higher – in *forte*. Since samples may have their amplitudes normalised, additional amplitude scaling can be imposed to allow finer control over dynamics. In this manner both qualities related to dynamics – sound level and timbre – can be affected simultaneously with one control parameter – MIDI velocity.

Detached articulation is well reproduced by multisampling. Any required articulation may be recorded, and replayed on demand. Sampling, though, does not have effective means to reproduce fluent note transitions in articulations such as *legato*. It may use elements of subtractive synthesis and apply amplitude envelopes to adjacent samples in attempt to remove attack phases and mask a moment of sample transition. However, in order to produce convincing note transitions sampling needs to be supplemented with a more precise method in a transition region. Additive resynthesis or one of diphone synthesis variants may serve this purpose.

Multisampling may be a sort of solution to general lack of control over produced sound in sampling synthesis. However, even though full multisampling of pitch is easily manageable, adding subsequent controlled qualities makes number of samples grow exponentially. As an example, 42 clarinet pitches, each in twelve performance techniques[10], multiplied by four dynamics levels, and again multiplied by three variants of each sample to avoid repeatability, would make 6048 separate samples for one instrument only. In case of synthesizing an orchestral piece, about ten to twenty different instruments may be required. Not all pitch-dynamics-articulation combinations may be necessary, but still, it makes thousands of samples that need to be recorded and edited, so that they sound consistently. Not only producing extensive sample

[10] An incomplete list of clarinet articulations may consist of: long *vibrato*, long *non vibrato*, *staccato*, accented, four different trills, and four mordents.

libraries may require a tremendous amount of work and resources, but even managing and using them may be problematic. Studies were carried out to automate some parts of the management process using music information retrieval (MIR) tools [122].

Layering Samples

Sampling synthesis can utilise a variant of wavestacking, which may also be referred to as layering. It may be seen as a hybrid of sampling and additive synthesis, where instead of sinusoidal partials several sound samples are mixed, each with own amplitude envelope. If a synthesizer does not have such capability, it may be simulated by using overlapping key-zones in pitch or in velocity domain. For instance, a complementary, overlapping velocity layers allow to use key velocity to control proportions of a two-sample mix. Two, or three axis controllers allow to control larger mixes. Layers are often utilised to enhance main samples with effect sounds produced during instrument performance, such as key hit, pedal action, fret noise, breath, etc.

2.2.2.4. Control of Duration

Sound samples contain separate sound events characterised by specific duration and evolution of some kind. In comparison to synthesis methods that utilise periodic or noise generators, able to produce signal for as long, as required, control over duration of reproduced sound samples requires more effort. Two possible solutions had been already applied in analogue samplers, and their evolved variants are still in use in digital synthesizers. The first one is to use long recordings, and stop them on request. The second one involves reproducing signal in a loop.

The first solution, a long recording, has the advantage of preserving natural signal evolution. A sample length is limited by either capability of the original signal source, such as breath of a wind instrument performer, or by a technical solution utilised in particular synthesizer. There is always a possibility, that even a long recording may not be sufficiently long. The Mellotron utilised strips of magnetic tape that contained approximately eight seconds of recording, and it was often too short. An interesting workaround was invented in a form of 'crawling spider' playing technique. Every time a tapes were nearing end, a performer was changing inversion of played chord, with palm movement resembling aforementioned arachnid. Contemporary software samplers may impose no length restrictions, but source-related restrictions are still valid. One may argue, that e.g. if oboe cannot play a note such long, then a sampler replaying oboe sound shall also not. But in some cases such possibility might have been useful.

The second solution, a looped playback, addresses the length issue, allowing to produce any required duration through repetition of a sample or its segment. In analogue looping samplers, e.g. employing optical discs, the whole sample content was looped. Consequently, a sound contained no natural attack phase, since only a section with sufficiently flat amplitude envelope, taken out of the middle of the recording, could have been utilised. Since attack is one of key dynamic timbre characteristics [263], its absence in samples resulted in worse resemblance to original sounds.

To preserve attack, in digital sampling only a section of sample is looped. This section is specified by **loop points**, and should be located after the attack, where sound level is relatively stable, and before the release phase (Fig. 2.36). When sample is reproduced it continues through start loop point L_1 up to end loop point L_2. From L_2 it jumps to L_1, and repeats section L_1–L_2 for as long, as the key is pressed. When the key is released, it continues from the actual position to the end of sample.

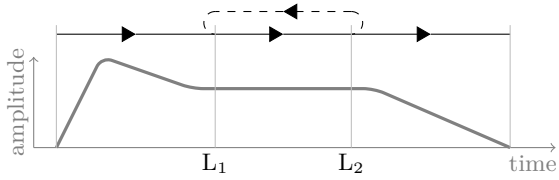


Figure 2.36. Placement of loop points L_1 and L_2 in an idealised sound sample envelope

Even though looping solves a duration issue, it may result in undesired, audible effects. In order to remain possibly seamless, loop points need to be carefully chosen. It may be carried out automatically, or manually [470]. Russ points out selection criteria that can be summarised as follows (Fig. 2.37) [485]:

- signal value has to be equal in both loop points to prevent discontinuity,
- first derivative of signal has to be approximately equal on both ends to assure continuous signal slope.

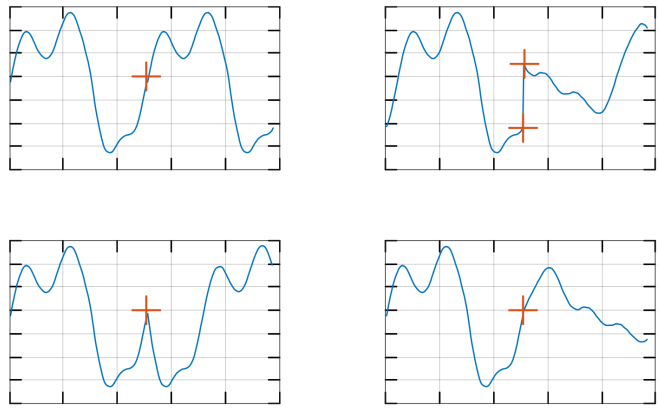


Figure 2.37. Choice of loop points and resulting splice effect: correct (top left), discontinuous signal values (top right), switched sign of first derivative (bottom left), changed value of first derivative (bottom right)

Equal signal value condition is often fulfilled simply by choosing loop points in signal zero-crossings. Signal period shall also be considered, so that a loop contains

integer number of periods. Roads [470] and Howell [253] discuss some algorithms and tools that automate selection of loop points. Early digital samplers used very short loops, lasting only a few periods, to limit memory usage. Such loops, however, with all natural evolution removed from a sustain phase, resembled fixed-waveform wavetable synthesis. Without strict memory limits, current sampling synthesizers often use loops lasting several seconds [253].

A loop points splicing issue may be attenuated by applying a different looping method. A simple one is to perform a bidirectional loop (Fig. 2.38b). A loop continues forwards from point L_1 to L_2, then goes backwards to L_1, and so on. Another variant reproduces bidirectional loop in two simultaneous layers (Fig. 2.38c). In one a loop is reproduced forwards, in the other – backwards. A splice point may also be masked by applying crossfade (Fig. 2.38d). End of loop gradually fades out, while beginning fades in.

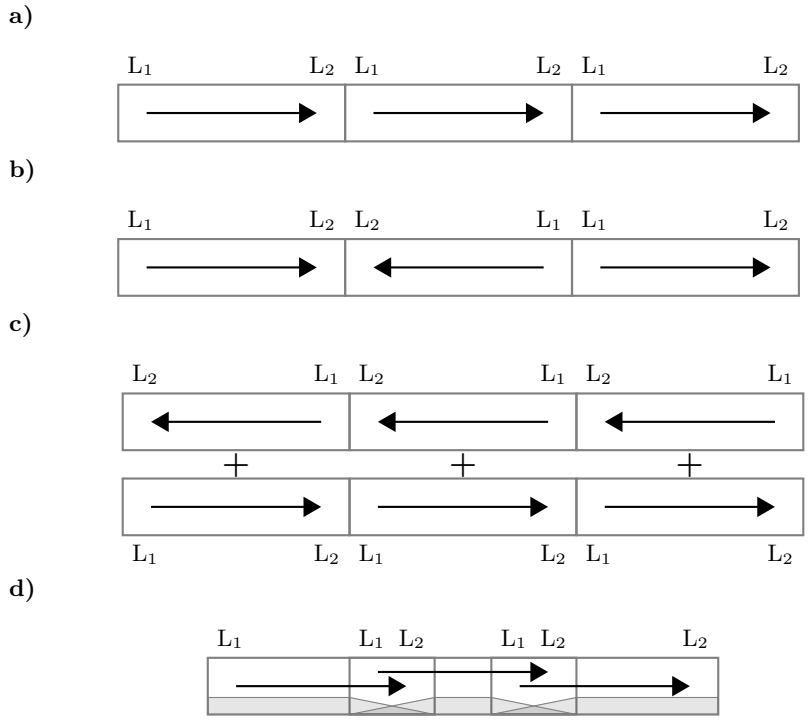


Figure 2.38. Looping methods: a) simple splicing; b) bidirectional loop; c) bidirectional layered loop; d) crossfaded loop; on all diagrams time goes from left to right, and arrows mark direction of loop reproduction
Source: author's elaboration, based on Roads [470]

Looping may distort a natural signal variability, such as *vibrato*, or slow buildup of tension. Longer loops can make it less apparent, but still, on long notes even longer loops will be repeated several times, and this periodicity may be audible. The problem

reqiores more refined solutions, such as time-granulation or additive resynthesis. Once sustain phase of a sample has been analysed, and is reproduced from partials, its duration may be controlled in the same manner, as in additive synthesizer [470].

2.2.2.5. Application of Envelopes and Filters

Apart from the most basic samplers, sampling synthesizers utilise EGs that control amplifier and low-pass filter. Even though such set of tools could be applied to impose a new, artificial evolution on sample contents, under normal circumstances its purpose is different. It provides means to simulate articulation and dynamics-related changes of timbre, such as softening the attack phase, without a need to utilise massive multisampling.

Howell [253] discusses a procedure that can be applied to simulate dynamics-related timbre changes in struck or plucked string instruments (Fig. 2.39). The procedure assumes that dynamics is controlled through MIDI key velocity value, which is a standard approach. It uses three mechanisms:

- skipping initial sample segment of adjustable length,
- applying amplitude envelope with adjustable attack segment,
- applying low-pass filter.

Since all these mechanisms result in removal of some parts of signal, either in time or frequency domain, an original sample to start with needs to represent the most rich variant of instrument sound. A loud and bright sound, with distinct attack transient is usually a good choice.

The first step involves removing part or all of attack transient. It is carried out through mapping key velocity to a number of initial signal samples skipped, so that low velocity values remove more of the attack section, and the highest possible velocity removes none. It may be calculated according to the following formula

$$t_S = f_S(127 - V) \tag{2.46}$$

where t_S is the time skipped, which needs to be translated into a number of samples considering sampling rate, $V \in [0, 127]$ is the key velocity, and $f_S(\cdot)$ is some monotonically increasing function satisfying the condition $f_S(0) = 0$. It may be linear, e.g. $f_S(x) = ax$, where a is the scaling factor controlling the maximal skipped part.

Skipping an initial sample fragment removes a part of plucking or striking sound. However, it has a side effect of possible click, or at least makes the beginning unnatural. An objective of the second step of the procedure is to produce artificial attack in place of removed natural one. For this purpose an amplitude envelope is imposed. Its attack time is controlled by the velocity value. Again, the highest velocity should make the attack instantaneous, allowing all the recorded natural attack to be played without attenuation, and lower velocity should produce longer attack, making it softer. A formula similar to (2.46) can be applied

$$t_A = f_A(127 - V) \tag{2.47}$$

where t_A is the duration of the attack phase in the imposed envelope. Function $f_A(\cdot)$ needs to satisfy the same conditions the f_S does, but in general both functions may be different.

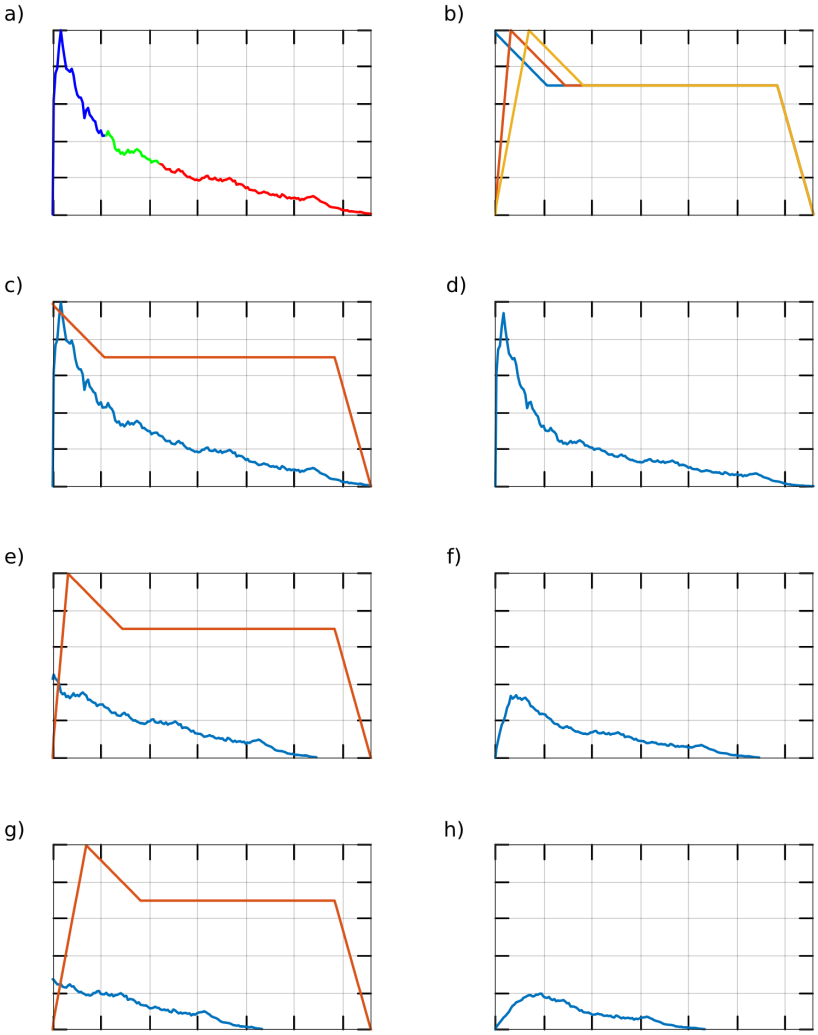


Figure 2.39. A simulation of various dynamics levels; dynamics is controlled through MIDI key velocity value by skipping an initial sample part (a) and applying amplitude envelope (b); plot (c) presents the maximal velocity case, and plot (d) the result – no skipping is applied, and imposed attack is instantaneous; plots (e) and (f) present medium velocity, with blue section skipped, while plots (g) and (h) – low velocity, with blue and green sections skipped

Source: author's elaboration, based on Howell [253]

The final step involves applying a low-pass filter with a cut-off frequency depending on key velocity. A constant cut-off frequency or a cut-off controlled by the same envelope that controls an amplitude may be applied. For the highest velocity a filter should pass all the signal, but lower velocities shall produce less bright timbre through lowering cut-off frequency.

Functions f_S and f_A depend on sampled instrument, or even on a particular sound sample. Obviously, the procedure is not universal, and some instruments may require a different approach, although applied properly, it may allow other timbre qualities to be simulated.

Advanced sampling synthesizers may provide more signal modifying facilities, inherited from subtractive method, such as a number of independent LFOs, and more flexible modulation matrix. They may be applied to compensate for lack of signal evolution in looped sustain phase. Outputs of separate LFOs may be directed to signal amplitude and filter cut-off frequency. Setting appropriate value of LFO delay value will prevent modulation from affecting an attack phase.

2.2.2.6. Sampler Features and Implementation Remarks

Analogue sampling-based instruments were simple replay devices, aimed at substituting larger ensembles or difficult to carry instruments. Even after transition to digital domain, sampling was initially considered as an additional function, accompanying other, more sophisticated synthesis methods. Such was a case of early models of Fairlight CMI in 1970s. Yet, when memory became less expensive and more capacious, the main factor hindering quality of samples had gradually vanished. Availability of large libraries containing realistic samples of virtually any useful instrument or effect sound (Tab. 2.5) allowed sampling to become, and continue to be a synthesis method of choice in music arrangement. As a curiosity, sampling is applied not only to reproduce acoustic instruments, but also sounds produced by electronic instruments and other synthesis methods.

Sampling synthesis has two crucial advantages. The first is low complexity, resulting in very modest requirements for processing capabilities. The second is an ability to produce reasonably realistic simulation of some popular instruments. Even though complexity may seem not to matter in case of software synthesizers operating on fast computers, it is still, actually, a key factor in music production. If massive, multi-voice musical pieces have to be synthesized in real time, and only a fraction of processing unit time may be diverted to synthesis, due to rest of it being consumed by processing audio data in a large digital audio workstation project, the more efficient synthesizer allows to work on more complex projects.

Nevertheless, in comparison to other synthesis methods, sampling is seriously limited in its control capabilities. Throughout its development as a method it has been addressed by adapting many elements from other, more flexible methods – particularly from subtractive synthesis. Many contemporary samplers might be considered subtractive synthesizers, only with an oscillator replaced as a signal source by an audio file. Other than that, all the subtractive facilities, such as envelopes, LFOs, filters, and modulation matrix, are present.

Table 2.5. Contents of sample libraries

Content	Description
Solo instruments	Grand piano, guitar, etc.
Orchestral instruments	Either single orchestral instruments or groups, like strings, woodwinds, brass, etc.
Ethnic and exotic instruments	Crwth, hurdy-gurdy, oud, etc.
Standard band instruments	Rock, jazz, pop, etc.
Drum kits	Acoustic and synthetic
Synthesizers and electronic	Vintage and modern
Voices and choirs	Different voices, various vocal ensembles
Drum loops	Rhythmic sequences
Phrases and loops	Various musical excerpts
Ambient textures	Noises, people talking, nature background, etc.
Foley and other effects	Steps, household sounds, engines, explosions, etc.

It may, though, still be difficult to attain level of control over produced signal comparable to what is possible in subtractive synthesis. The problem is natural evolution inherent in recorded samples. It manifests itself through irregular modulations as well as slow variations of parameters that may be attributed to envelopes controlling a timbre. In subtractive synthesis oscillator signal is stationary, and variability is imposed through modifiers, making a result predictable. In sampling, intrinsic source variability combined with artificially imposed evolution renders the effect virtually unpredictable in all but the simplest cases. In practice, if the advantage of realistic samples is to be exploited, signal modifications have to remain subtle. Thus they are often limited to applying ADSR or similar amplitude envelope, and enabling low-pass filter.

Commercial sampling-based synthesizers are sometimes differentiated into 'synth', 'sampler', and 'hybrid' groups, depending on provided features. The groups, though, often intersect, as particular synthesizers gain new functions in subsequent releases. Generally, most synthesizers tend to accumulate functions over time, and aim towards multi-method synthesis engines. Only small fraction remain implementations of a pure, straightforward sampling method.

Since one of main applications of sampling method is music arrangement and related tasks, commercial synthesizers in a software plug-in form aim at high efficiency. Most of them replay samples directly from non-volatile memory, such as hard (HDD) or solid state (SSD) drives. HDDs and SSDs have much larger capacity than RAM, therefore tens of thousands of samples may be utilised. Indeed, sample libraries bundled with current sampling synthesizers occupy 50–70 GB of storage. Even with such numbers, if samples utilised at the moment are cached in RAM, it allows to play hundreds of simultaneous voices.

Contemporary samplers utilise various sample-replay techniques. Even though most support multisampling, they also allow to transform samples in time and frequency domain. Time-stretching and pitch-shifting is carried out either by simple sample conversion, or through more advanced resynthesis-based methods, such as time granulation or phase vocoder, which allows to correct pitch-shifting related formant structure deformations. Besides sample transformations, various looping and splicing modes are usually available. As odd as it seems, high quality of samples may be a problem, since it makes sample repeatability clearly apparent, if only one sample is used for a particular key-velocity combination. As a remedy, samplers utilise a number of alternative samples that are randomly chosen when the same note is played repeatedly. Another quality-related function implemented in a number of samplers is an emulation of 'vintage' digital samplers, i.e. a possibility to simulate the effect of lower sampling frequency, low bit-rate, or artifacts produced by various types of early digital to analogue converters.

Majority of sampling synthesizers implement multisampling technique, accompanied by flexible sample assignment routines, such as configuration of key-ranges with key and velocity overlapping, as well as layering. Due to multisampling, sample libraries may contain tens of thousands of samples. An advanced sampler aimed at simulating a guitar alone, able to reproduce slight variations in playing technique, can utilise as much as 9500 samples. With such libraries sample management becomes an issue. Most samplers implement a tree structure to organise library contents. Yet, if a library does not have key and controller assignments for all sample variants factory-set, or if a user attempts to prepare own sample library, a tremendous amount of work might be required. Therefore a number of samplers introduce some procedures that automate parts of this process. Pitch-detection algorithms are utilised to automatically map samples to appropriate keys, and define valid key-zones. An interesting feature is an automatic extraction of multisamples from VST instrument plug-ins, where a sampler controls another virtual instrument, playing and sampling its sounds, according to initial user decisions, such as sample duration, or number of sampled velocities.

Even though detailed modulating of recorded samples is problematic due to applying modulation to already modulated signal, advanced commercial synthesizers utilise source-modifier approach of subtractive synthesis, with comparable set of modulation modules. They include envelope generators, LFOs, and step sequencers, as well as possibility to modulate a sample with external signals, e.g. incoming MIDI. Modulators are directed through modulation matrix to various signal modifiers. Apart from filters, sampling synthesizers often include large sets of effects, such as reverbs and delays, distortions and bit crushers, or dynamics processors, that can be arranged in effect chains. Due to high efficiency of sampling method, it is often possible to separately process each voice.

A new feature introduced in later releases of major sampling synthesizers is scripting. It is aimed at customising tasks that can be carried out automatically. A very simple example would be a programmable *arpeggiator*. Scripting may provide a way to effectively exploit multi-samples and modulation modules, particularly if a number of samples gets larger, and their manual assignment becomes unmanageable. Script-

ing is concentrated on particular use scenarios. A script may help to produce more realistic sequences. As an example, by selecting appropriate samples and controlling their envelopes a legato phrase may be synthesized, which is otherwise a very difficult task for sampling synthesizer. Some scripts may be instrument specific and simulate string selection, chord voicing, alternating bowing, strumming, or switching between a single instrument and an ensemble. Different kind of scripts may recognise a musical context, and e.g. harmonise a melody, or impose context-specific micro-tuning. Composition-oriented scripts may produce generative or algorithmic music, or perform music transformations, such as mirror, inversion, or transposition. In case of a few samplers scripts are written in their own, proprietary languages. Yet, there are also samplers that can be scripted in general, open scripting languages, such as Lua [260].

Due to improvements and elements adapted from other synthesis methods, sampling synthesis has overcome or lessened some of its crucial weaknesses. In its refined, complex forms it may be a very realistic and convincing resynthesis tool, able to reproduce sound of a number of instruments almost perfectly, and produce satisfactory results in case of many others. Yet, this has been attained through application of very large sample libraries combined with signal modifiers that require numerous parameters tuned to individual samples. Scripting helps to automate much of scenario-specific tasks involving sample selection and processing, but even if using scripts may not be difficult for end user, writing them requires much more expertise, and may prevent many users from going beyond scripts that are factory-provided.

Therefore control capabilities still remain a serious disadvantage, and if they are of key importance, one should utilise other synthesis methods. A flexible method that provides an interesting set of control abilities, and is still based on waveforms, is the granular synthesis. On the side of realistic reproduction, sampling has still issues with instruments that involve continuous control, such as solo violin or cello. It is particularly audible in fluent phrases. This problem, in turn, is addressed by concatenative synthesis, that in many aspects may be considered a descendant of sampling.

2.2.3. Granular Synthesis

Even if one looks at music as a purely acoustical phenomenon, without considering its contexts and meanings, it consists of various levels of diverse structures in time and frequency domains. It is interesting then, that majority of sound synthesis methods aim at one of these levels only, and produce sound events representing single musical notes, just like most of traditional musical instruments that are controlled at this exact level. In twentieth century ‘a note’ had already been losing its significance and place as a basic structural element in numerous musical styles. Since sound synthesis is not bounded by the limitations of traditional instruments, and may be controlled in entirely different ways, it might be used to generate different levels of musical structures. Such is the case of granular synthesis, which may be considered either a set of audio signal processing techniques, a sound synthesis method, or a composition technique.

In granular synthesis sound, in a form of objects or events, is produced through arrangement of sonic grains (Fig. 2.40), defined by Roads as brief, microacoustic events [472]. A grain has a duration that nears auditory limits for perception of its time and frequency-based characteristics – usually from 1 to 100 ms. Spectral synthesis methods are frequency-based. Sampling or wavetable synthesis are time-based. While the grain is basically also time-based, actually it is considered a convenient sound representation due to combining time-domain and frequency-domain information (Tab. 2.6).

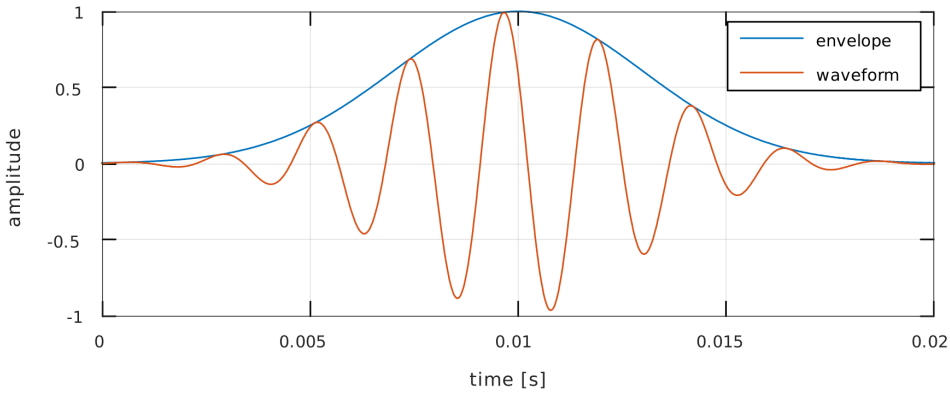


Figure 2.40. A single grain with 440 Hz sinusoidal waveform and Gaussian envelope

Table 2.6. Information contained within a grain, according to Roads [472]

Time-domain	Frequency-domain
Starting time	
Duration	Pitch
Envelope shape	Spectrum
Waveform shape	

The concept is based on works of physicist Dennis Gabor and composer Iannis Xenakis. Gabor postulated [197, 198] that granular representation may describe any sound, and designed granular mechanism for independent time-stretching and pitch-shifting. The claim was verified by Bastiaans [38, 39, 40]. Granular sound representation was also proposed by Wiener [606] and Moles [382]. Windowing, a key element of controlling a grain, was applied in the short-time Fourier transform (STFT) [497]. Xenakis applied granular sound theory in composition, producing grains by the means of analogue tone generators and tape splicing [611]. Curtis Roads was the first to implement granular synthesis in a digital form, using a computer [468].

2.2.3.1. Grains

A sonic grain is a short fragment of sound shaped by amplitude envelope. Any signal origin is possible, either synthetic or sampled. In case of the former, grains are usually built out of harmonic partials which results in a fixed waveform, or are produced through some modulation technique resulting in a time-varying signal. As for the latter, grains are sampled from selected positions within a sound file, and may be pitch-shifted. In Figure 2.40 the signal is a 20 ms long section of 440 Hz sinusoid, and its amplitude is shaped by a Gaussian envelope.

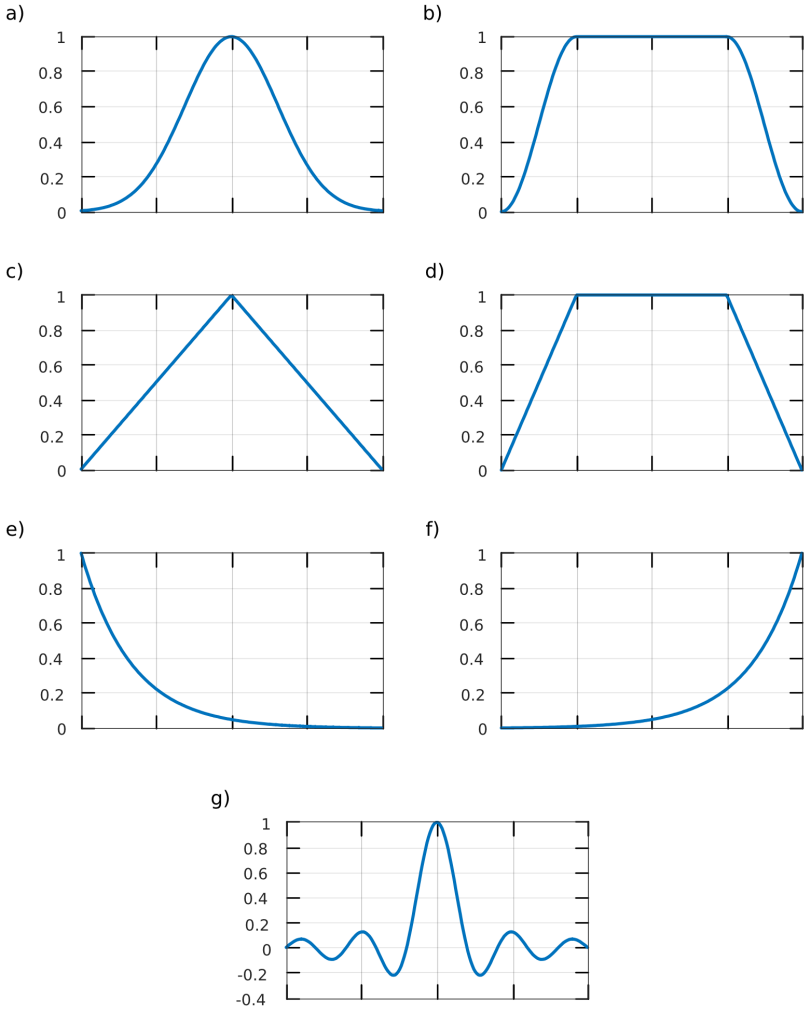


Figure 2.41. Envelopes used in sonic grains: a) Gaussian (2.48) with $\sigma = 0.31$; b) Tukey (2.49) with $\alpha = 0.5$; c) triangular (2.50); d) trapezoid (2.51) with $\alpha = 0.5$; e) expodec (2.52) with $\tau = \frac{N}{6}$; f) rexpodec (2.53) with $\tau = \frac{N}{6}$; g) sinc (2.54) with $\tau = N/10$

Perception of a single grain depends on its duration. According to Roads [472], weak impression of pitch appears in grains longer then 5 ms. Above 25 ms a grain has a distinct pitch and timbre of a waveform it is based on. If duration is 2 ms or lower, it is perceivable as a click, and its timbre is strongly affected by envelope, though waveform properties still have some effect. Grain duration may be fixed, random, or related to a selected parameter, such as frequency. Even small variations in duration within large clouds of grains affect resultant timbre significantly.

Grains can be produced using almost any envelope, though only several envelopes have their properties and effects studied in greater detail (Fig. 2.41). Functions used in some envelopes extend to infinity, and need to be truncated. In early works of Gabor [197] envelopes were assumed to be **Gaussian**. Such envelope in a discrete form is expressed as [110]

$$w[n] = e^{-\frac{1}{2}\left(\frac{n-(N-1)/2}{\sigma(N-1)/2}\right)^2} \tag{2.48}$$

where $\sigma \leq 0.5$, N is the envelope length in samples, and n is the sample index.

In case of larger grains amplitude envelope controlled by **Tukey window** allows longer section of waveform to remain unattenuated, since it convolves cosine lobes with rectangular window. It is also referred to as **tapered cosine window** or **quasi-Gaussian envelope**, and assumes the following form [231]

$$w[n] = \begin{cases} \frac{1}{2}\left(1+\cos\left(\pi\left(\frac{n}{n_1}-1\right)\right)\right) & \text{for } 0 \leq n < n_1 \\ 1 & \text{for } n_1 \leq n \leq n_2 \\ \frac{1}{2}\left(1+\cos\left(\pi\left(\frac{n}{n_1}-\frac{2}{\alpha}+1\right)\right)\right) & \text{for } n_2 < n \leq (N-1) \end{cases} \tag{2.49}$$

where $n_1 = \frac{1}{2}\alpha(N-1)$, $n_2 = (N-1)(1-\frac{\alpha}{2})$, and $\alpha \in [0,1]$.

If computational efficiency is vital, line-segment envelopes may be applied, although they introduce some additional spectral effects compared to Gaussian or Tukey envelopes [291]. **Triangular envelope** may be calculated according to [231]

$$w[n] = 1 - \left|\frac{n-\frac{N-1}{2}}{\frac{N}{2}}\right| \tag{2.50}$$

while **trapezoid envelope** is expressed as [600]

$$w[n] = \begin{cases} \frac{n}{n_1} & \text{for } 0 \leq n < n_1 \\ 1 & \text{for } n_1 \leq n \leq n_2 \\ \frac{N-1-n}{N-1-n_2} & \text{for } n_2 < n \leq (N-1) \end{cases} \tag{2.51}$$

where $n_1 = \frac{1}{2}\alpha(N-1)$, $n_2 = (N-1)(1-\frac{\alpha}{2})$, and $\alpha \in [0,1]$.

Roads mentions two asymmetric exponential envelopes [472], one decaying, referred to as **expodec**

$$w[n] = e^{\left(-\frac{n}{\tau}\right)} \tag{2.52}$$

and the other being a reversed variant, or **rexpodec**

$$w[n] = e^{\left(\frac{n-N+1}{\tau}\right)} \tag{2.53}$$

where τ should be related to N. Finally, sinc function, or **band-limited pulse envelope**

$$w[n] = \text{sinc}\left(\frac{n-\frac{N-1}{2}}{\tau}\right) \tag{2.54}$$

where τ needs to be related to N, produces grains with a 'resonant' characteristic [470].

A list of parameters that may be controlled on a basis of a single grain include duration, envelope, waveform frequency, relative amplitude, spatial location, and waveform itself. The last one is a set of sound production parameters for synthetic grains, or a file and a location within it for sampled grains. Roads points out that since granular synthesis trades off instrument complexity for score complexity [472], grains may be produced by a simple synthesizer, such as one presented in Figure 2.42, although more complex solutions can be applied as well.

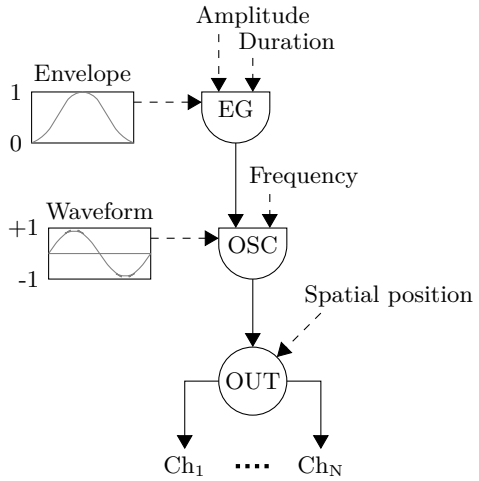


Figure 2.42. A simple grain generator with a single-cycle wavetable oscillator as a source of waveform; grains have controllable duration, frequency, amplitude, and spatial position – through a choice of output channel; various envelopes and waveforms may be chosen
Source: author's elaboration, based on Roads [472]

A single sonic grain requires a relatively large number of parameters. Considering that average number of grains per second, also referred to as grain density, may reach several thousands [470], it is impractical to control all of grains parameters manually and directly. Control data reduction is achieved through organising grains into higher-level units. Such units require a reasonably small number of global parameters, and in turn control all parameters of grains within.

There are six types of granular synthesis, characterised by different **organisation of grains** [472]:

- time-frequency plane matrices and screens,
- pitch-synchronous overlapping streams,
- synchronous and quasi-synchronous streams,
- asynchronous clouds,
- physical and abstract models,
- granulation of sampled sound.

2.2.3.2. Time-Frequency Plane Matrices and Screens

Matrices originate from analyses of existing sounds, performed using the Gabor matrix [472], the STFT, or the wavelet transform (WT). Analysed signal is transformed from time-domain to a two-dimensional quantised matrix on a time-frequency plane.

The Gabor transform is a windowed analysis technique that relates time-domain signal of effective duration Δt to a frequency-domain spectrum of effective spectral width Δf [197, 198]. Such 'acoustic quantum' is bounded by an uncertainty relation [197]

$$\Delta t \Delta f \geq 1 \tag{2.55}$$

and may have either good temporal or spectral resolution, but not both at the same time. Gabor defined quanta as [197]

$$g(t) = e^{-a^2(t-t_0)^2} e^{2\pi i f_0 t} \tag{2.56}$$

where the first part is the Gaussian envelope, the second – the complex sinusoidal function, and parameter a determines the uncertainty [197]

$$\begin{aligned} \Delta t &= \pi^{\frac{1}{2}} a^{-1} \\ \Delta f &= \pi^{-\frac{1}{2}} a \end{aligned} \tag{2.57}$$

In case of STFT matrices grains are aligned on a time-frequency grid with linear frequency scale and fixed duration of analysis window. A single grain may comprise a set of overlapping windows in each analysis channel [19]. WT matrices differ from STFT in logarithmic frequency spacing and variable analysis window duration that is a function of frequency [310]. Both kinds of matrices may be used to perform analysis, transformation, and resynthesis of recorded sounds such as pitch-shifting or time-stretching. They may be regarded as general signal processing methods rather then sound synthesis.

Xenakis proposed a concept of screens [611]. A screen is a Gabor matrix at a given moment, in an amplitude-frequency grid. Sound evolution emerges through reproducing a synchronous sequence of screens, referred to as 'a book'. The technique is aimed more at synthesis alone than at analysis and resynthesis, therefore screen contents may be produced by the use of practically any generative algorithms. Screens may be filled by random scattering of grains, and further subjected to set-theory operations. Xenakis postulated, that such operations lead to production of new sounds, not related to these produced by traditional instruments or physical objects. The other example are screens produced by cellular automata [78, 371, 372, 373].

2.2.3.3. Pitch-Synchronous Granular Synthesis

The aim of pitch-synchronous granular synthesis (PSGS) is to reproduce pitched sounds with spectra containing formant regions [156]. It is an analysis and resynthesis method, therefore it allows to transform recorded sounds [107].

An initial stage involves spectral analysis, where individual time-frequency cells represent grains. Each cell has its frequency response as well as fundamental frequency estimated. Frequency response is calculated as filter coefficients at each cell boundaries on frequency axis. Fundamental frequency is determined at cell boundaries on time axis [470].

In resynthesis stage a pulse train with detected frequency is produced and filtered through a bank of parallel minimum phase finite impulse response filters with previously estimated coefficients. Therefore pulse train excites weighted sum of all filters impulse responses. Grains neighbouring in time domain are overlapped (Fig. 2.43a) to produce smooth signal variations [470].

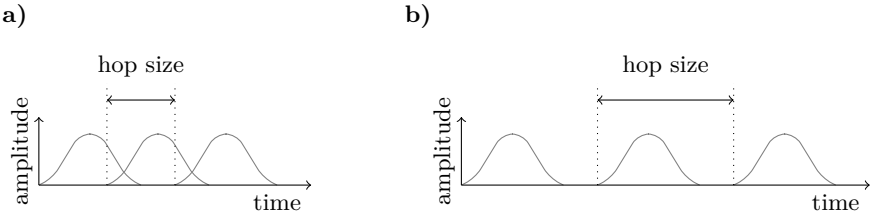


Figure 2.43. Stream of grains: a) overlapped; b) spaced apart

Similarly to improvements that LPC brings over vocoder in subtractive synthesis, there are PSGS extensions that can separate quasi-harmonic and residual inharmonic parts of sound [436].

2.2.3.4. Synchronous and Quasi-Synchronous Granular Synthesis

Grains can be organised into one or more streams. Such technique is exploited in synchronous granular synthesis (SGS) as well as quasi-synchronous granular synthesis (QSGS). Stream is a simple granular organisation, where grains are produced consecutively, and time interval between them is important for characteristics of output signal. In case of SGS and QSGS (Fig. 2.44) the term 'synchronous' refers to regular

time intervals. Stream parameters are relatively easy and intuitive to control, e.g. using a computer program with graphical user interface [417].

SGS can produce pitch or rhythmic effects, depending on grain density. For densities lower than 20 grains per second, and grains spaced apart (Fig. 2.43b), the effect is rhythmic. A gradual change of density leads to *accelerando* or *rallentando* effect. Higher densities produce pitched sounds with strong fundamental frequencies. Through appropriate selection of grain duration values and envelopes, sidebands may be introduced and controlled. They can be perceived either as separate pitches, or form a formant peak.

Roads notes [472], that perception of pitch in SGS may not be apparent. It depends on interactions of three periodicities: period of waveform within a grain, period of envelope determined by grain duration, and period determined by grain density. Under different circumstances one of these may dominate the others, or none of them may dominate and pitch may be ambiguous. In very dense streams SGS may even be perceived as noise, with synchronicity effect imperceivable.

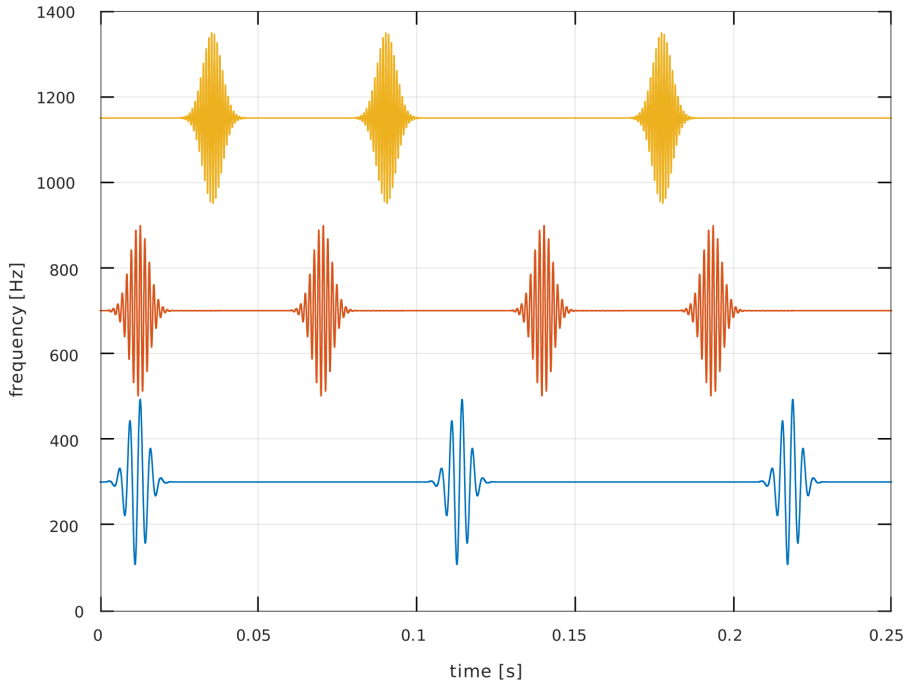


Figure 2.44. Streams in quasi-synchronous granular synthesis; frequency of grain waveform is indicated by placement of stream on vertical axis; in all streams grain duration is 40 ms; blue stream has an average time interval of 100 ms, red – 60 ms, and yellow – 80 ms; interval irregularity has the highest value in yellow stream
Source: author's elaboration, based on Roads [470]

In QSGS time intervals slightly vary from grain to grain (Fig. 2.44), therefore output signal may be considered a case of amplitude modulation, with grain envelope being the modulator. Each sinusoidal carrier component modulated by periodic envelope function produces a series of sidebands at a frequency distance equal to inverse of envelope period. Sidebands amplitude is controlled through a grain envelope. Due to modulation formant regions are produced instead of single frequency components. QSGS allows to blur formant structure in a controllable way by making time interval between grains less regular [562, 563]. Blurred formant structures lead to 'thicker' sound textures [470].

2.2.3.5. Asynchronous Granular Synthesis

Just as SGS and QSGS utilise streams of grains, asynchronous granular synthesis (AGS) organises sonic grains into clouds. Clouds are regions of time-frequency plane, onto which grains are scattered using stochastic or chaotic algorithms. Roads compares using AGS to a spray jet, where each dot in the spray is a sonic grain [469]. Clouds are defined with a set of parameters collected in Table 2.7.

Table 2.7. Cloud specification parameters, according to Roads [470, 472]

Parameter	Description
Start time	For the entire cloud
Cloud duration	As for a lower bound, clouds may be as short as 20 ms
Grain duration	May be constant or random within limits, may follow a curve, or may vary in function of grain frequency
Density of grains	Can vary within a cloud; high density produces overlapping grains
Frequency band	Typically defined by two curves that define low and high frequency boundaries (*cumulus* clouds); it is also possible to restrict grains to a defined set of pitches (*stratus* clouds)
Amplitude envelope	For the entire cloud
Waveforms	Can vary between grains within a cloud; sampled and synthetic grains can be mixed
Spatial dispersion	Usually implemented as a number of separate output channels

Duration of grains impacts the bandwidth of the output signal – shorter grains produce wider spectra (Fig. 2.45). If waveform within a grain is synthetic, it can be controlled with a compact set of parameters, such as amplitudes of partial sinusoids in case of additive synthesis. Otherwise, i.e. in case of sampled waveforms, control is more limited and involves choosing a recording and position at which playback of a grain will start. Roads names three kinds of clouds regarding a choice of waveform [470]. **Monochrome** clouds consist of a single waveform type, **polychrome** randomly mix several waveforms, and **transchrome** 'mutate' between a number of waveforms.

Clouds can also be differentiated according to their shapes on a time-frequency plane (Fig. 2.46). It must be noted though, that in this plane frequency applies to

grain waveforms only. Spectrum of output signal may significantly exceed limits set by cloud shape due to effects of grain envelopes or grain density. Common cloud shapes are: *cumulus*, *stratus*, and *glisssandi*. In cumulus clouds grains are scattered randomly between two curves that form a frequency band. By narrowing a band output signal gains more definite pitch. Instead of bands, stratus clouds occupy only a limited number of specific frequencies, and can form chord-like structures. If these frequencies change over time, such clouds are referred to as glissandi.

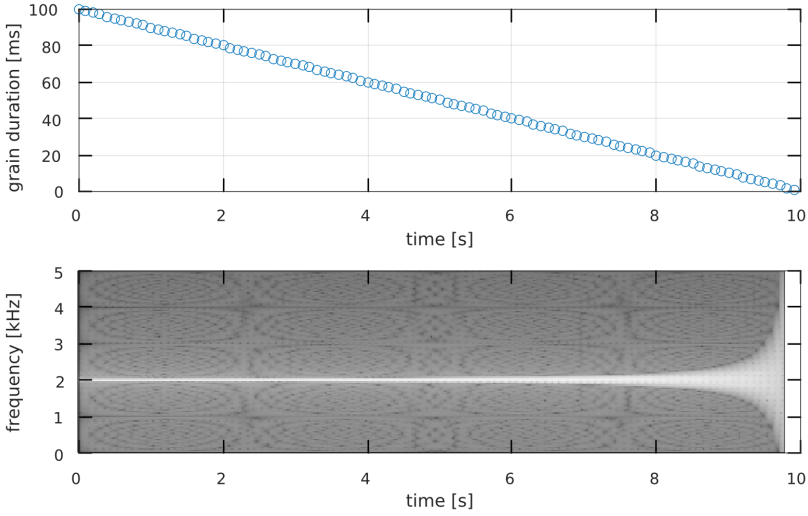


Figure 2.45. Impact of grain duration on output signal spectrum; signal was produced using grains with sine waveform and Gaussian envelope; grain waveform frequency was fixed at 2000 Hz, and grain density was 10 per second; grain duration was changing according to the top plot; resulting spectrogram is presented in the bottom plot

Texture of a cloud is determined not by grain duration or density alone, but by a combination of both – a **fill factor**, defined as [472]

$$FF = T_g \rho_g \tag{2.58}$$

where T_g is the grain duration in seconds, and ρ_g is the grain density, i.e. number of grains per second. In AGS both parameters may vary within a cloud, hence FF is estimated using their average values. For FF values below 0.5 a cloud is considered **sparse**, around 1.0 it is **covered**, and above 1.0 it becomes **packed**. However, due to random placement and overlapping of grains in AGS clouds, $FF = 1.0$ does not guarantee a solid cloud – there may be a number of silence sections left. Solid clouds are produced with FF values greater then 2.0 [472].

Granular clouds may be considered as sound objects, and form musical building blocks. Such objects are controllable on a grain level as well as on a global cloud level, providing a flexible means to create and organise musical structures, with interesting

evolution effects. Individually synthesized clouds may be treated as separate sound events and mixed in a traditional way. A different approach is also possible. Instead of synthesizing individual clouds, the whole musical form may be composed and synthesized in a granular domain – thus AGS may transcend a simple means of sound production, and can become a formative element.

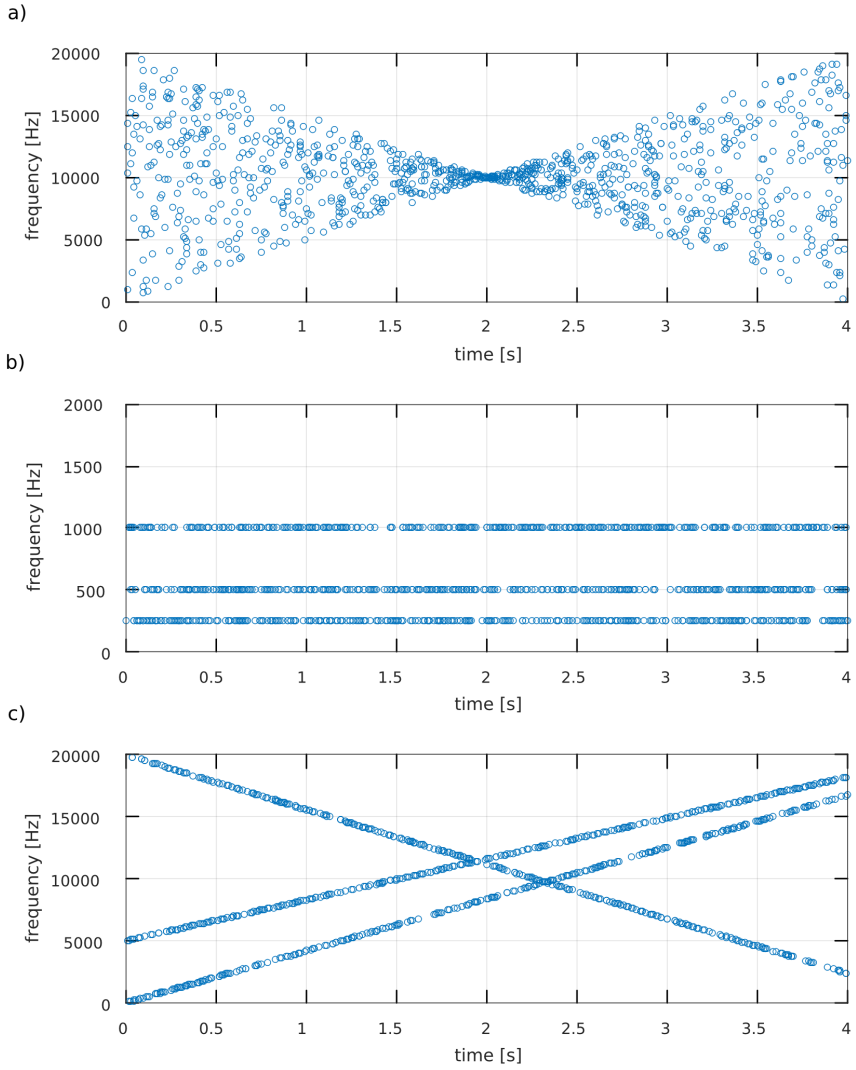


Figure 2.46. Cloud shapes in asynchronous granular synthesis: a) cumulus; b) stratus; c) glissandi
Source: author's elaboration, based on Roads [470]

2.2.3.6. Physical and Algorithmic Models

Many sounds produced by physical objects may be regarded as granular, i.e. consisting of discrete sound events in a microscale. Such sounds are characteristic for moving particles enclosed in resonant containers, or for scraped objects. Examples of instruments that produce sounds falling into this category would be maracas or güiro – the former shaken, the latter scraped. Granular synthesis provides an efficient way to simulate such kind of sounds without a requirement to design a complete numerical model, and yet allows to control musically and physically meaningful parameters of a simulated object.

Physically informed sonic modelling (PhISM) is a general synthesis approach proposed and implemented by Cook, [134, 135, 136], where understanding of the physical system behaviour is used to produce sound through some efficient synthesis methods, without actually simulating its entire vibrational behaviour. If sound is produced through granular synthesis, the method is referred to as physically informed stochastic event modelling (PhISEM). Grains are controlled through parameters obtained from physical observations and measurements, simulations, and heuristics. Since numerical simulations are often easier to carry out and provide more immediate results then the actual measurements of physical systems, they are the usual data source for further synthesis process. In case of maraca, data collected include frequency, amplitude, and time regarding collisions of beans with the enclosure. During synthesis it is adequate to utilise a white noise with exponentially decaying envelope as a signal source, and shape it through a set of resonance filters that simulate resonance frequencies excited in collisions of particles with distinct instrument elements (Fig. 2.47). Cook presented PhISEM solutions for a number of percussion instruments, and objects such as coins. Other studies present algorithms for different instruments and objects, e.g. metallic ball, or stream of water. In the former case, the authors refer to the method as the *ecologically-based granular synthesis* [292]. Similar models can be created to simulate sound of various mechanical systems under changing conditions – necessary data is available in a number of publications within a filed of mechanics [218, 346, 348, 365].

Algorithms governing particle characteristics and behaviour do not need to simulate physical objects or systems – they can be based on other principles as well. One of more popular approaches is to arrange grains using chaotic functions [241, 384, 207]. Contrary to stochastic, random number based arrangements that – unless some additional, shaping function is imposed – produce simple structures that are either uniform and uni-directional, or mirror some other statistical distribution, such as Gaussian or Poisson, chaotic functions easily switch from stable to unstable states [161, 209, 210, 162, 472]. Such behaviour can be exploited to dramatically change the character of produced sound with small parameters adjustments. However, designing mappings between synthesis parameters and chaotic behaviour that produce useful and interesting musical effect may be non-trivial.

Interesting effects can be obtained through sonification, i.e. translation of data into sound. Data can have any origin, but in granular synthesis many attempts have been made to sonify various physical phenomena outside of classical mechanics, that do not have natural connection to sound – from quantum mechanics, to astrophysics.

Xenakis in GENDYN system experimented with trajectories of particles bouncing off elastic barriers [611, 509].

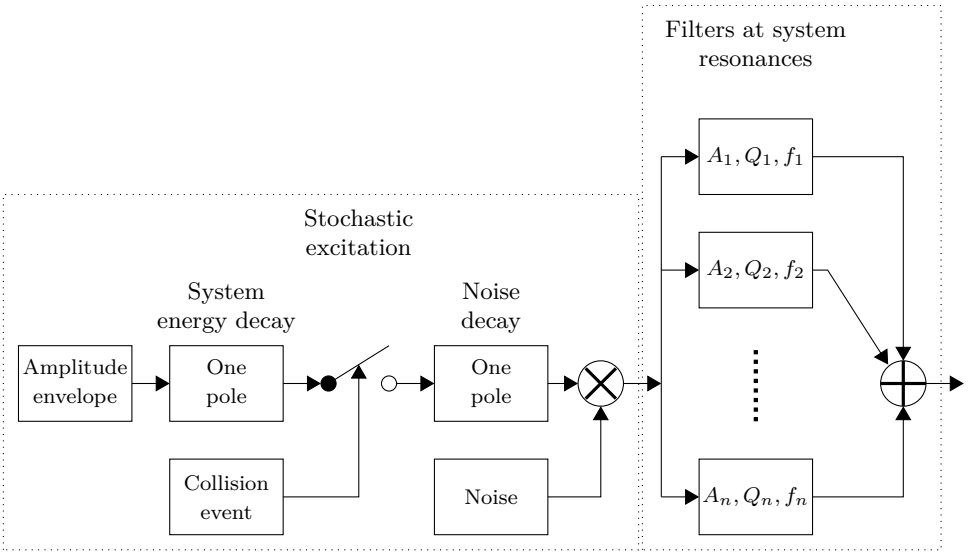


Figure 2.47. Basic PhISEM algorithm
Source: author's elaboration, based on Cook [136]

2.2.3.7. Granulation of Sampled Sounds

The technique is also referred to as time-granulation of recorded sounds. It concerns various ways of handling sampled sound using granular approach. In the first step small sections of sound sample are read either from file, or directly from an analogue-to-digital converter (ADC) [562, 563]. Next, envelopes are applied, to produce grains. And finally, grains are emitted in a specified order. If granulated signal comes directly from ADC, means to reorder grains is limited, but in case of sound files any order is possible.

For grains sampled from a file, three simple scenarios are the most common (Fig. 2.48). In the first one, only a single, large grain is extracted, and cloned in a sequence – such is the case of e.g. a snare drum roll. Another scenario involves producing many grains from a single file, and emitting them in changed, and usually random order. The last case is similar, but grains are produced using not one, but more sound recordings.

If grains are sampled from an ADC, two approaches are possible. In the first one granulation provides a sort of digital delay line. In the second, playback speed may be flexibly slowed down through repetition of selected grains.

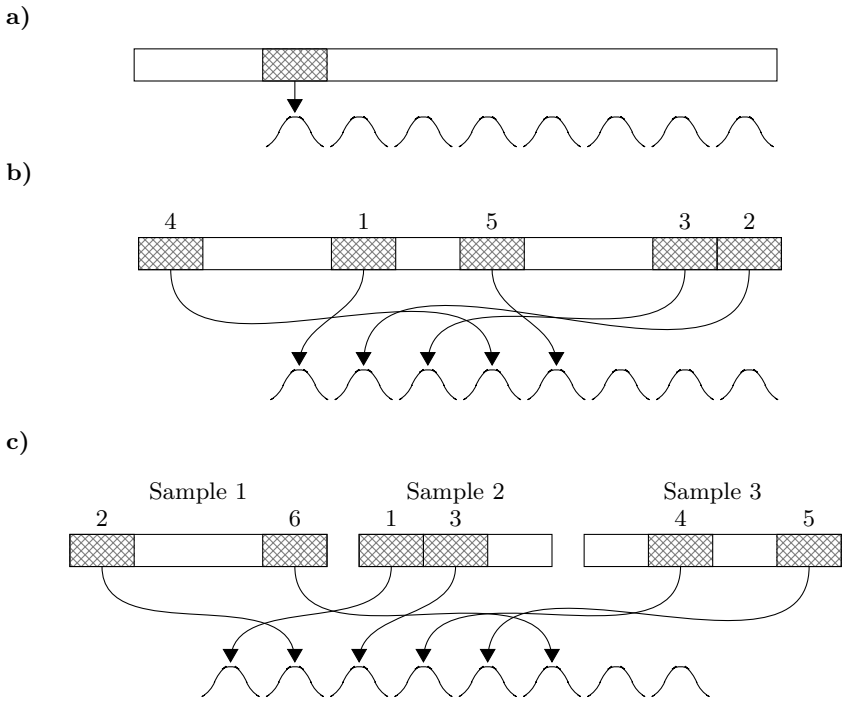


Figure 2.48. Granulation of sampled sound stored in a file: a) replication; b) reordering; c) merging and reordering
Source: author's elaboration, based on Roads [470]

2.2.3.8. Particle Synthesis

Methods such as PSGS, SGS, QSGS, and AGS may be considered the core of granular synthesis, but there also exists a number of methods that are either derived from basic granular synthesis, or are based on similar principles. Roads categorises them as particle synthesis methods [472].

Glisson synthesis introduces a concept of *glisson* as an enhancement of a sonic grain. Glisson is a grain which frequency changes either up or down, producing a short *glissando* effect. It may be represented by a vector describing its trajectory on a time-frequency plane. Glissons are grouped in clouds, where each of them can have a different trajectory. In this regard glisson synthesis is analogous to AGS, and has a similar set of control parameters. However due to vector nature of its particles it introduces additional property, referred to as *magnetisation pattern* [472], that controls trajectories of glissons within a cloud (Fig. 2.49).

In wavelet synthesis grain duration is related to its frequency, i.e. the higher the frequency, the shorter the grain. This relation can be generalised, and any granular synthesis parameters may be linked in a similar manner. Such technique is referred to as **grainlet synthesis** [472]. It may resemble modulation matrix utilised in subtrac-

tive synthesis, but with grain and cloud parameters instead of parameters of source and modifiers. In simple cases a parameter may influence another one directly, or inversely. In more complex scenarios a *point of attraction* may be defined. For instance, while linking frequency and duration, grains may become longer when their frequencies get closer to such point.

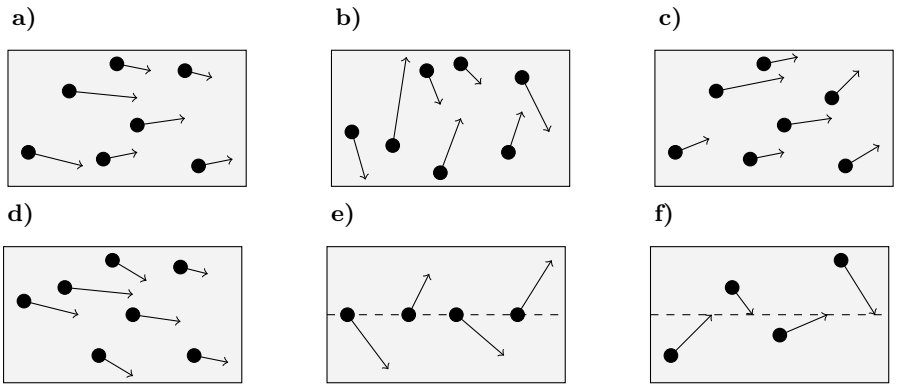


Figure 2.49. Glisson synthesis magnetisation patterns: a) bidirectional shallow; b) bidirectional deep; c) upwards unidirectional; d) downwards unidirectional; e) diverging from center frequency; f) converging to center frequency; horizontal axis represents time, vertical – frequency
Source: author's elaboration, based on Roads [472]

Another variant of a sound particle is the *trainlet* – a short series of pulses. **Trainlet synthesis** organises trainlets into clouds [472], similarly to other granular methods. Trainlets introduce additional grain-related parameters, such as pulse period, its harmonic structure, or spectral energy profile. Analogue pulse trains may be filtered to obtain stipulated spectra. Digital, on the other hand, may already be produced according to specified spectral parameters. Typical digital implementations involve bandlimited harmonic pulses [607, 252, 535] with spectrum controlled through *chroma* parameter [472]

$$A_k = A_0 \chi^k \tag{2.59}$$

where k is the harmonic number, χ is the chroma, A_0 is the amplitude of the fundamental, and A_k is the amplitude of the k-th harmonic. Chroma may assume any value – positive, or negative, as well as non-integer. For $\chi = 1$ spectrum envelope is flat. For $\chi > 1$ lower, and for $\chi < 1$ higher harmonics are attenuated. Therefore, larger values correspond to brighter sound timbre (Fig. 2.50). Trainlet clouds exhibit inherent evolution of parameters, which are defined as pairs of values, initial and final. Cloud starts with a parameter set to the initial value and gradually morphs towards the final value. A list of trainlet synthesis parameters is presented in Table 2.8.

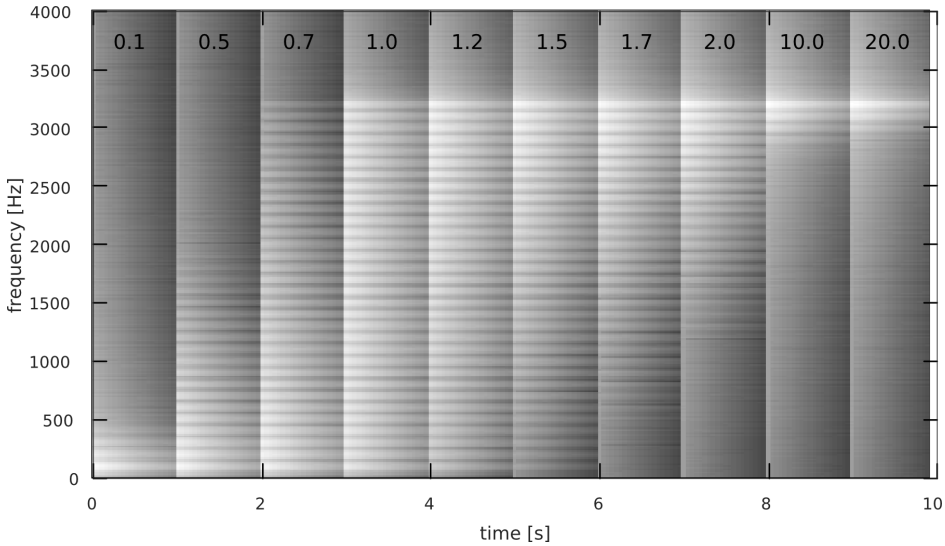


Figure 2.50. Impact of chroma (values inscribed in top part of plot) on spectra for exponentially decaying harmonic signals ($f_0 = 100$ Hz, 32 partials); for each value of chroma signal amplitude has been normalised separately

Table 2.8. Parameters of a trainlet cloud, according to Roads [472]; 'I/F' denotes a pair of initial (at the beginning of a cloud) and final (at the end) values

Parameter	I/F	Typical value
Cloud start time	—	—
Cloud duration	—	—
Trainlet durations	X	0.001–0.8 s
Random trainlet duration flag	—	—
Density of trainlets	X	1–300 per second
Upper frequency bandlimit of the cloud	X	20–20000 Hz
Lower frequency bandlimit of the cloud	X	20–20000 Hz
Cloud sound level	X	−96–0 dB FS
Number of harmonics	X	1–64
Lowest sounding harmonic	X	—
Chroma	X	—
Waveform	X	usually sine
Spatial position of trainlets	X	—
Attack time	X	5–50 ms

Pulsar synthesis (PS) may seem similar to trainlet synthesis due to utilisation of pulse signals, yet it is much more powerful and flexible. It was inspired by astronomical objects – rotating neutron stars or white dwarfs that emit a beam of electromagnetic radiation. Due to rotation a beam is observed only while pointed at Earth, which results in a pulsed appearance. Observed pulsar periods range from seconds to milliseconds [11], therefore the phenomenon covers time scale of rhythmic and tonal events.

In PS a sound particle is referred to as *pulsar* (Fig. 2.51), and it has two segments: *pulsaret* and silence [472]. A pulsaret is an arbitrary waveform, such as band-limited pulse, sine, multicycle sine, decaying multicycle sine, or a sampled signal. Total pulsar duration p is a sum of two values [471]

$$p = d + s \tag{2.60}$$

where d is the duration of pulsaret, and s is the duration of silence interval. Therefore p represents a pulsar period and d corresponds to a pulse width of a pulse signal. Repeated pulsar produces *pulsar train* that has two frequency parameters: $f_p = p^{-1}$, or a fundamental frequency, and $f_d = d^{-1}$, which may be referred to as a formant frequency, because it produces the effect of a formant peak in signal spectrum. Typically f_p values fall between 1 Hz and 5 kHz, and f_d values between 80 Hz and 10 kHz [472]. Both frequencies are continuously variable and controlled by separate envelopes (Fig. 2.52). It is possible for pulsaret duration d to exceed total pulsar duration p. In such case pulsaret may be truncated, with optional crossfading, or may be overlapped, which produces phase cancellation effects.

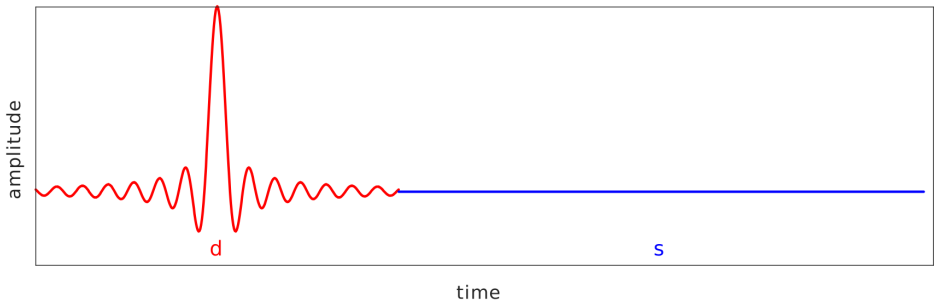


Figure 2.51. A single pulsar; the initial red section is a *pulsaret* with duration d, and the following blue flat section is a silent interval with duration s; in this particular case pulsaret is a band-limited pulse signal, but other types of signal may also be utilised
Source: author's elaboration, based on Roads [471]

[11] The first discovered millisecond pulsar, PSR B1937+21, rotates with 1.557708 ms period, corresponding to almost 642 Hz.

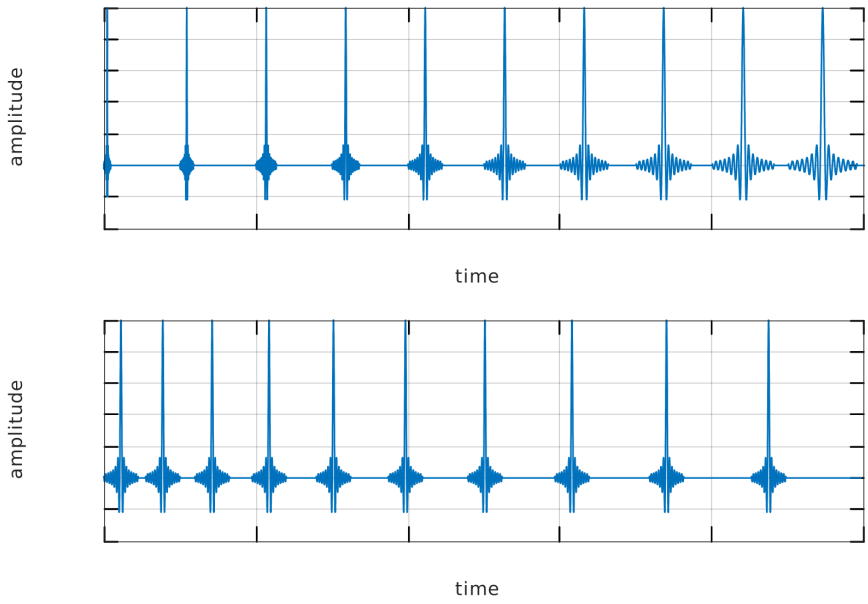


Figure 2.52. Evolution of a pulsar train; in the top plot pulsar period remains constant, and pulsaret duration changes; in the bottom plot pulsaret duration is fixed, but pulsar period changes due to increasing silence section
Source: author's elaboration, based on Roads [471]

Another signal characteristic that can be controlled in PS is the envelope of a pulsaret, which may differ from rectangular, affecting signal spectrum. E.g. a three-segment envelope is characteristic for formant synthesis techniques such as FOF or Vosim.

PS may be enhanced in several ways. Pulsar train from a generator may be partially masked, with selected pulsarets muted, resulting in a specific pattern – either deterministic or stochastic. When f_p is within a rhythmic range, it results in a rhythmic sequence. Otherwise, muted pulsars introduce subharmonic partials resulting from local lengthening of pulsar period. Different PS enhancement involves using multiple pulsar generators working with common f_p, but with separate values of f_d. Such arrangement produces a signal with several formants, where each generator controls a single formant through its f_d value. Finally, pulsar trains may be convolved with sampled sounds.

Particle synthesis is well-suited to produce spectrum with conveniently controllable formant regions. Roads mentions three such methods [470, 472]: formant wave-function (fr. *fonction d'onde formantique*, FOF), Vosim, and window-function (WF) synthesis. The first two were initially aimed at speech synthesis, while the last one attempted to simulate formants of acoustic instruments.

FOF synthesis may be considered a variant of PS with a specific grain envelope. In FOF a stream of damped-sine grains produces a spectrum with a formant region. Time-interval between grain onsets determines fundamental frequency of a signal, and grain envelope parameters control formant shape. Another, global envelope controls the entire sound event. A FOF grain envelope is produced using the following expressions [472]

$$\begin{aligned} w_A(t) &= \frac{1}{2}\left(1 - \cos\left(\frac{\pi t}{t_{tex}}\right)\right) e^{-t_{atten}t} \quad 0 \le t \le t_{tex} \\ w_D(t) &= e^{-t_{atten}t} \qquad\qquad\qquad\qquad\qquad\quad t > t_{tex} \end{aligned} \tag{2.61}$$

where w_A and w_D is the attack and decay segment of the envelope, respectively, t_{tex} is the attack time, and t_{atten} is the decay time. The most important FOF parameters control formant central frequency ($p1$), its -6 dB bandwidth ($p2$), peak amplitude ($p3$), and the lower part of peak – a *formant skirt* at -40 dB below the peak. Parameter $p2$ is determined by the grain decay time, and $p4$ by the grain attack time. Several FOF generators are used in parallel to produce signal with more formant peaks. Bennett and Rodet presented various implementations of FOF, including synthesis of musical instruments [49]. An implementation that simplifies control over multiple FOF streams was proposed by Rodet at al. [477]. Although possible, FOF-based resynthesis is difficult [36, 450, 31, 601].

In **Vosim** synthesis sound particles are decreasing pulse trains, where each pulse is a squared sine [276, 277, 548, 278]. A single particle contains N_p such pulses. The first pulse within a train has amplitude A, and amplitudes of the following pulses decrease by a decay factor b. Parameter T_p is a duration of each pulse, and a train is followed by a variable delay N_d. Therefore, fundamental period of a signal produced by Vosim synthesis may be expressed as [472]

$$T_V = N_p T_p + N_d \tag{2.62}$$

where T_V determines fundamental frequency, and T_p determines formant peak position. Fundamental frequency may be controlled without affecting formant position by changing delay N_d, and by modulating it a *vibrato* effect or noise may be introduced. Similarly to PS and FOF, each formant region requires a single Vosim generator. Additional parameters may control type of frequency modulation, maximum frequency deviation, or modulation rate. Signal may evolve if N_p, T_p, N_d, or A changes over time.

The last of particle formant methods, the **window-function** (WF) synthesis produces a stream of grains separated with silence intervals, thus it may be considered a variant of PS. The effect is a broadband and harmonic signal which requires further processing – a *weighting stage* – to attenuate or emphasize selected harmonics. The grain is based on a *window function pulse*, which is a special waveshape that has stipulated spectral properties. In initial works Bass and Goeddel [37, 208] have chosen Blackman–Harris window function, which has a low level of side lobes, therefore it is virtually band-limited and produces very little aliasing.

The window in a symmetric, 4-term variant (Fig. 2.53) is expressed as [231]

$$
\begin{aligned}
w[n] &= a_0 - a_1 \cos\left(\frac{2\pi n}{N-1}\right) + a_2 \cos\left(\frac{4\pi n}{N-1}\right) - a_3 \cos\left(\frac{6\pi n}{N-1}\right) \\
a_0 &= 0.35875 \\
a_1 &= 0.48829 \\
a_2 &= 0.14128 \\
a_3 &= 0.01168
\end{aligned}
\tag{2.63}
$$

where N is the window size, in samples. Other window functions may be applied. Like in the PS, in the WF sonic grain a pulse is followed by a silence interval referred to as a *deadtime*. Fundamental frequency of a signal produced is controlled by varying deadtime duration, while the pulse duration remains unchanged (Fig. 2.54). For lower values of f_0 number of harmonics increases – more harmonics fit within the center lobe of the window function spectrum. Due to increasing duration of deadtime for decreasing f_0, WF requires frequency-related amplitude compensation.

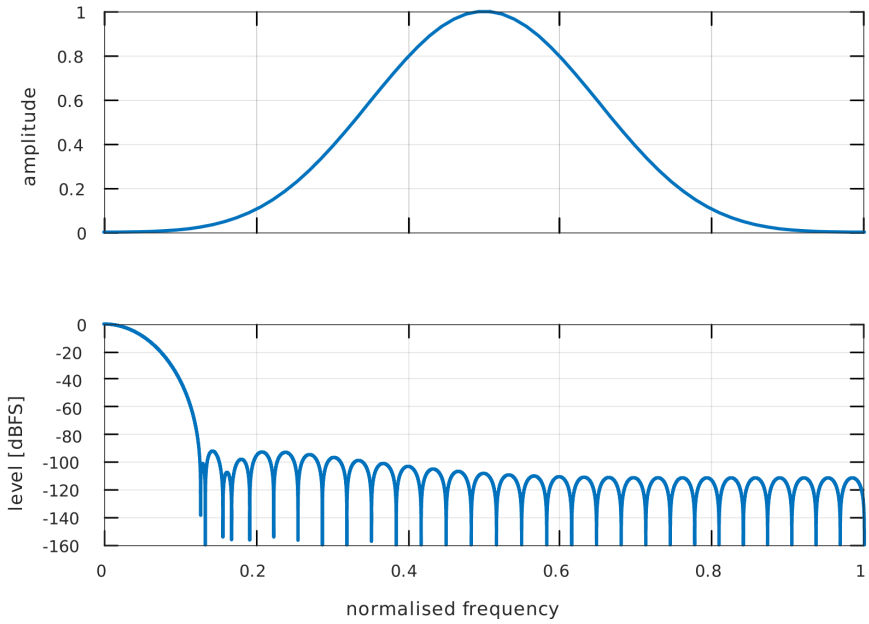


Figure 2.53. Blackman–Harris window – a basis of WF pulse: waveshape (top), and spectrum (bottom); only half of the spectrum is plotted – value of 0 represents the center frequency of the spectral peak

Further processing varies amplitudes of harmonics, producing formant regions through *slot weighting*. A *time slot* is a single pulse and the following deadtime – it

corresponds to a single pulsar in PS. In weighting process amplitudes of time slots are periodically multiplied by a sequence of *slot weights*. Spectrum of a stream of weighted pulses has a formant structure with peaks and valleys.

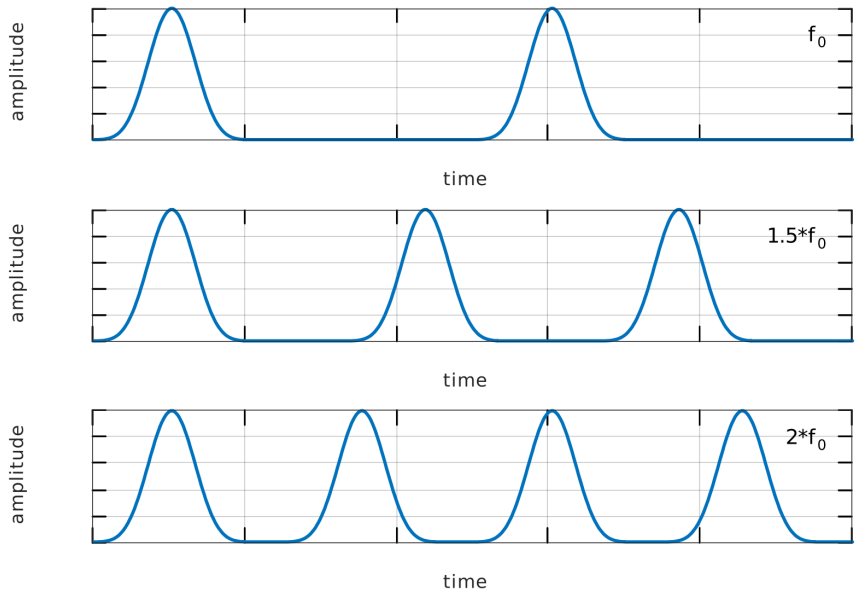


Figure 2.54. Change of signal fundamental frequency in WF synthesis; pulse width remains constant, while deadtime interval varies; longer deadtime in lower frequencies results in lower level of output signal, and thus requires amplitude compensation

Many other synthesis or sound processing methods may be classified as particle methods. Apart from above mentioned methods, Roads describes a number of particle synthesis methods with possible musical applications, including **particle cloning synthesis**, **synthesis by transient drawing**, or various sonification-related techniques [472].

2.2.4. Concatenative Synthesis

From a musician's standpoint a concatenative synthesis (CS), inspired by a speech synthesis method [297, 452], is an attempt to solve note-transition related issues of sampling synthesis that prevent it from producing fluent musical phrases. It may be considered a kind of enhancement to the sampling method, and indeed, in some works – presumably due to its genesis – it is categorised as a sub-group of sampling synthesis [123]. However, even though compared to sampling CS is relatively new, it is a more general method, and thus it may be placed systematically on a higher level.

According to Schwarz [503] in its general formulation CS encompasses sampling as well as granular synthesis.

CS methods can be applied on different levels, and use a variety of techniques [500, 501, 502, 503, 513, 102, 504, 505, 344, 56, 255, 410, 411]. Therefore, in its general form CS is defined using common abstract terminology that in particular implementations refers to elements sharing some minimal set of common properties, but otherwise representing different classes of objects and algorithms. In some implementations CS is also referred to as mosaicing synthesis [619, 324, 269, 124, 123].

A starting point of CS is a database of source sounds (Fig. 2.55) [503]. Since one database may be utilised for various synthesis systems, particular synthesis uses its subset referred to as *corpus*. Sounds within a database are time-segmented into *units*. A sound or a musical phrase that is to be synthesized is referred to as a *target*. A *unit selection* algorithm finds a sequence of units that best matches the target on the basis of unit *descriptors*. Descriptors may be either extracted from units and describe their characteristics, or attributed to them and contain some higher-level information. The units selected to match the target may not fully match it. In such case they need to be transformed. Afterwards, they are concatenated to produce an output signal.

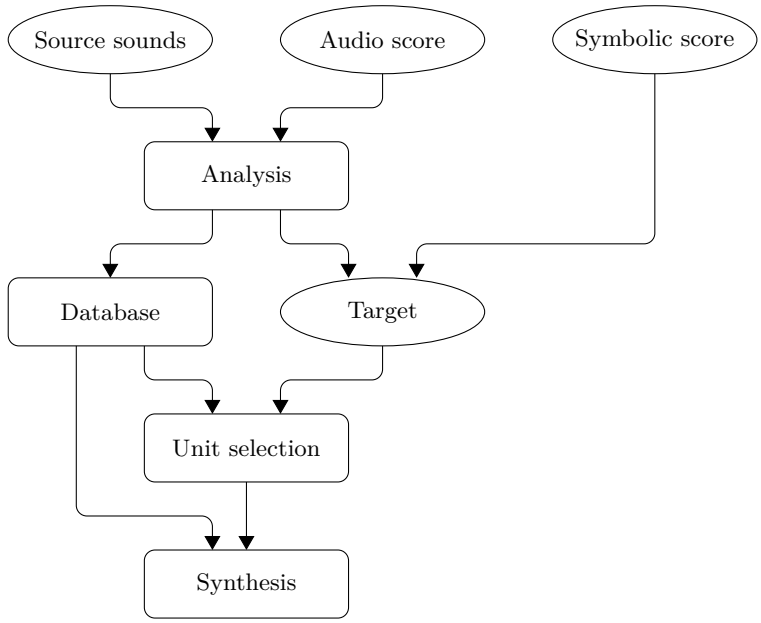


Figure 2.55. A general structure of a concatenative synthesis system on the basis of Caterpillar system presented by Schwarz
Source: author's elaboration, based on Schwarz [501, 502]

CS is an example of a content-based processing paradigm, where operations performed on sound are based not on signal processing alone, but also on symbolic or high-level data [14]. Other examples based on the same paradigm, and dealing with

problems similar to CS, are context-sensitive effects [337], manipulation of the audio signal basing on its musical structure [270], the Song Sampler system for automatic sampling of parts of songs [24], or the MusEd software for browsing through songs using pitch and loudness descriptors [122].

In various implementations CS rules may be either supplied, or induced from the sound data in a *data-driven* approach [500, 502]. Usually both sources are utilised, in various proportions. Units may represent various levels of sound objects: from snippets, through single notes, to phrases, and they need not to be uniform.

Concatenative synthesis is not as universal as more traditional synthesis methods, such as additive, subtractive, or wavetable. However, there are particular applications where it may be targeted to bring significant improvements. Schwarz [503] lists its three main applications: high-level instrument synthesis, resynthesis of audio, and free synthesis. Moreover, even though CS may seem as a method suitable only for assembling musical phrases on the basis of previously known score, a real-time implementation has also been developed (Fig. 2.56).

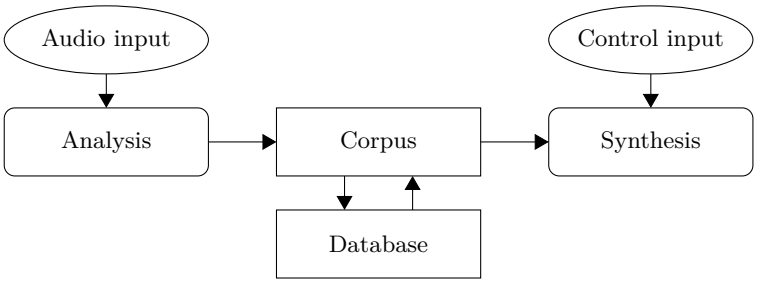


Figure 2.56. Structure of a real-time corpus-based concatenative synthesis; audio input is feeding the corpus; corpus may be saved and loaded from database in a persistent manner; synthesis retrieves data from the corpus
Source: author's elaboration, based on Schwarz et al. [504, 505]

The first application is the most obvious, and CS is particularly well suited to serve as a source of instrumental parts reproducing entered musical score, for instance during work of a composer or in musical arrangement tasks. Due to ability to store context information regarding sounds within its database, it can reproduce context of target score and assemble musical phrases with natural sound transitions, which is an issue in basic sampling synthesis. Moreover, details not covered in score, such as phrasing-related local parameter irregularities, are filled in by database units.

The second application is less obvious. There are synthesis methods that allow to resynthesize a single sound of a particular instrument, thus allowing to reproduce it while altering some of its characteristics. CS is able to do this for the entire recorded performance. A recording of a phrase, referred to as an audio score (Fig. 2.55) is subjected to a processing similar to database sounds when they are segmented into units. Next, it is resynthesized using units from database.

Finally, in free synthesis a heterogeneous sound database may be used directly. Descriptors provide a means for efficient and meaningful control over selection amidst

corpus of sounds. Thus free concatenative synthesis may be considered an extended granular synthesis with a number of additional features. One of advantages is the possibility to stipulate continuity through grain transitions, in any of the descriptors. Another advantage is the ability to search through a sound file in a meaningful descriptor space, and not simply by position.

2.2.4.1. Segmentation

Source sounds need to be segmented into units and analysed to obtain their characteristics in a form of descriptors. This process may be automatic, blind, or arbitrary. It may even be performed on-the-fly. Several different approaches can be utilised in a single application, as in the Caterpillar system [501].

The first approach is to utilise some external software. There is a number of implementations of algorithms performing, for instance, a note-segmentation [86, 588, 4]. Note-level units are commonly utilised by samplers. Another approach is possible if a musical performance contained within an audio file has a steady pulse, which may be determined by an appropriate algorithm [217, 181]. Such file can be segmented into regularly spaced units of equal durations. Segmentation of recordings that contain multitones requires additional effort to detect simultaneous pitches. It may be carried out on the basis of components asynchrony [616]. When a score is available for the audio file, it may be possible to go beyond a simple note segmentation and obtain a larger set of data for descriptors. It involves aligning a pitch-contour of the recording to the score using dynamic time-warping (DTW) algorithm (Fig. 2.57) [532, 279, 20, 280, 398, 99, 13]. Alternatively, score alignment may be performed using hidden Markov models (HMM) [454]. Once aligned, all the data contained within a score may be assigned to units as descriptors, including MIDI note number, lyrics, articulation, etc. As a final solution, it is possible to perform manual segmentation, or manually adjust result of automatic segmentation.

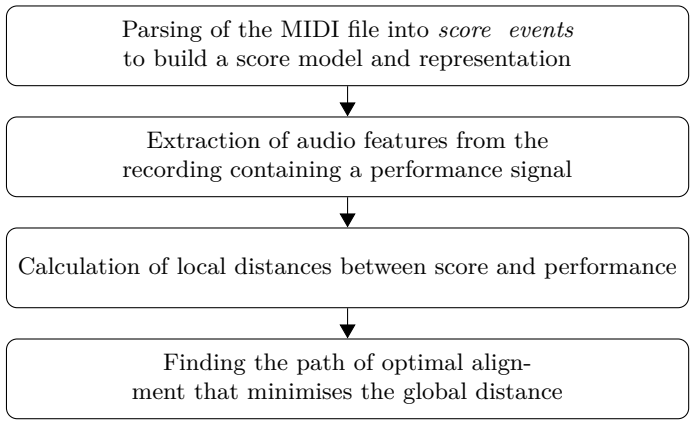


Figure 2.57. Steps of DTW-based procedure that aligns a recording to a score
Source: author's elaboration, based on Schwarz [502]

Automatic segmentation with a score alignment using DTW algoritm is a convenient solution. Comparing to other automatic approaches it is more flexible and provides more information to use in descriptors. Being score-based, it performs segmentation on a basis of pitch and produces units representing notes. It is, however, difficult to use pitch as a main feature, since pitch tracking algorithms are not universal, and all of them produce errors under certain circumstances, particularly in polyphonic performances. For the purpose of automatic segmentation a more reliable approach is to match a structure of peaks in harmonic spectrum. *Peak structure match* is defined as [502]

$$PSM(m,n) = \frac{\sum\limits_{i \in R(n)} S_i P_i}{\sum\limits_{i \in R(n)} P_i} \tag{2.64}$$

where m is the frame in the recording, n is the frame in the score, i is the FFT bin, S is the magnitude spectrum generated on the score basis, P is the Fourier magnitude spectrum of the analysed recording, and $R(n)$ is the frequency range for the score state n [502]

$$R(n) = \{i \in \mathbb{N} | i_{low}(n) \leq i \leq i_{high}(n)\} \tag{2.65}$$

Such operation may be considered a filtering with normalisation. DTW algorithm utilises an inverted structure, referred to as *peak structure distance* [502]

$$PSD(m,n) = 1 - PSM(m,n) \tag{2.66}$$

Using logarithmic scale for energy improves the procedure, due to reduced sound level differences between segmented notes as well as among simultaneously performing instruments. Generally, narrower filters are advantageous, since they allow to distinguish among more complex, dense harmonic structures. However, if the instrument analysed does not produce fixed pitch, like in wind or bowed string instruments, a certain degree of freedom around nominal filter frequencies is required [532]. Therefore, filters are set to pass a certain range of r cents around nominal frequency, and the energy passed is weighted with a Gaussian or Hamming window that penalises stronger, but more distant spectral peaks in favour of weaker, but closer to the nominal frequency.

Considering F_n analysed harmonic partials, the best tuning offset t for all filter bands k, where $1 \leq k \leq F_n$, of all notes p in score frame n, with nominal frequency $f(n,p)$ may be expressed as [502]

$$t_{max}(k) = \arg \max_{-r \leq t \leq r} \left(w_{hamming}(t) \sum_{i \in B_{n,p,k,t}} P_i \right) \tag{2.67}$$

where $w_{hamming}$ is the window function, and $B_{n,p,k,t}$ are indices of bins for the filter tuned [502]

$$B_{n,p,k,t} = \left\{ i \in \mathbb{N} \;\middle|\; \left| i \frac{f_s}{N_{FFT}} - k f(n,p) 2^{\frac{t}{144}} \right| \leq \beta_{n,p,k} \right\} \tag{2.68}$$

The harmonic filter band width given in tones is expressed as [502]

$$\beta_{n,p,k} = kf(n,p)2^{\frac{\beta}{6}} \tag{2.69}$$

where β is the basic filter width. As Schwarz points out [502], better results are obtained when filters are shifted independently, and not as a whole harmonic comb structure. It allows for a better fit to slightly inharmonic spectra of some acoustic instruments, with a notable example of piano [47, 191].

A local distance between a score frame n and a performance frame m is given by the *tuned peak structure distance* ($TPSD$) [502]

$$TPSD(m,n) = 1 - \log \frac{\sum\limits_{i \in B_{max}} P_i}{\sum\limits_{i \in R(n)} P_i} \tag{2.70}$$

where B_{max} is defined as a union of the best tuned filter bands in the analysed frame

$$B_{max} = \bigcup_{p,k} B_{n,p,k,t_{max}(k)} \tag{2.71}$$

Schwarz carried out a number of tests to determine parameters of the procedure that produce acceptable results [502]:

- $F_n = 6$ harmonics,
- filter width $\beta = \frac{1}{10}$ of a semitone (10 cents),
- tolerance $r = \frac{3}{4}$ of a semitone (75 cents).

The method described requires a refinement, since it produces alignment markers that are often delayed, and in cases with many simultaneous pitches skips some notes. The problem is related to attack phase, where energy is spread more evenly throughout the spectrum resulting in low PSD values. Reverberation makes partials of a previous note last after the next has begun, and sometimes a slow attack or window-related blurring causes energy in filters to reach its maximal values some number of frames after the note beginning.

The refinement is based on measuring a variation in the filters. It can indicate sharp rise in particular harmonics and, in consequence, beginning of the attack phase. At every onset special score frames are generated with energy variations instead of PSD values. Every band k of the $TPSD$ from (2.70) has its energy variation summed to calculate the attack distance (AD). If onsets are simultaneous, AD is calculated for every note and averaged, according to [502]

$$AD(m,n) = \underset{p}{\text{mean}} \left(1 - \tanh \left(\alpha \left(\sum_{k=1}^{F_n} |\Delta_k^p| - \theta_a \right) \right) \right) \tag{2.72}$$

where k is the index of harmonic filter, p is the note, Δ_k^p is the energy difference in dB with the previous local extremum in the band k, θ_a is the threshold, and α is the

scaling factor. Schwarz experimentally established the following values: $\theta_a = 6.5$ dB, and $\alpha = 50$ [502].

Notes may be followed by periods of silence. It is handled by adding special score frames at the end of each note. Special distance (SD) is a measure of the signal energy match above a silence threshold θ_s [502]

$$SD(m,n) = \begin{cases} E - \theta_s & \text{if } E \geq \theta_s \\ 0 & \text{if } E < \theta_s \end{cases} \tag{2.73}$$

where E is the total logarithmic energy of the signal

$$E = \log \sum_{i=1}^{N_{FFT}} P_i \tag{2.74}$$

Application of SD makes the alignment path to remain in the silence frame and advance in the recording.

Finally, the local distance matrix is built by combining all local distance models [502]

$$ldm(m,n) = \begin{cases} AD(m,n) & \text{if } n \in A \\ SD(m,n) & \text{if } n \in S \\ TPSD(m,n) & \text{if } n \notin A \cup S \end{cases} \tag{2.75}$$

where A is the first frame of a note and S is the last frame of a note.

DTW algorithm uses local distances as well as global constraints to estimate cost of the best path to point (m,n). It is expressed as *augmented distance matrix* $adm(m,n)$. Alignment is a path through adm, therefore a point (m,n) belonging to the path means, that a performance frame m is aligned with a score frame n.

Local path constraints, or neighbourhoods, are used to calculate $adm(m,n)$ from values of ldm. A number of local constraints types was discussed by Rabiner and Juang [455], but Schwarz tested types I, III, and V with Caterpillar system [502]. In type I

$$adm(m,n) = \min \left\{ \begin{array}{ll} adm(m-1,n-1) & + w_d ldm(m,n) \\ adm(m-1,n) & + w_v ldm(m,n) \\ adm(m,n-1) & + w_h ldm(m,n) \end{array} \right\} \tag{2.76}$$

where w_d, w_v, and w_h are weights of diagonal, vertical, and horizontal branches of the local path. In type III

$$adm(m,n) = \min \left\{ \begin{array}{ll} adm(m-1,n-1) & + w_d ldm(m,n) \\ adm(m-2,n-1) & + w_v ldm(m,n) \\ adm(m-1,n-2) & + w_h ldm(m,n) \end{array} \right\} \tag{2.77}$$

In type V

$$adm(m,n) = \min \left\{ \begin{array}{l} adm(m-1,n-1) + \lambda_{1,1} \\ adm(m-2,n-1) + \lambda_{2,1} \\ adm(m-1,n-2) + \lambda_{1,2} \\ adm(m-3,n-1) + \lambda_{3,1} \\ adm(m-1,n-3) + \lambda_{1,3} \end{array} \right\} \tag{2.78}$$

where

$$\begin{aligned} \lambda_{1,1} &= w_d ldm(m,n) \\ \lambda_{2,1} &= w_v ldm(m,n) + w_d ldm(m-1,n) \\ \lambda_{1,2} &= w_h ldm(m,n) + w_d ldm(m,n-1) \\ \lambda_{3,1} &= w_v ldm(m,n) + w_d ldm(m-2,n) + w_v ldm(m-1,n) \\ \lambda_{1,3} &= w_h ldm(m,n) + w_d ldm(m,n-2) + w_h ldm(m,n-1) \end{aligned} \tag{2.79}$$

Constraints of type I are sufficient for monophonic music. Types III and V are more robust in handling mismatches between score and performance. Schwarz points out that type V is preferable, giving the most freedom, although it uses the most resources [502].

The best alignment path is found using Viterbi algorithm [596], with the additional global constraints: alignment path starts in point $(1,1)$ and ends in (M,N), where M is the performance frame count and N is the score frame count, the path is monotonic, and the score is stretched to approximate duration of performance, i.e. $M \approx N$. Such constraints should produce path close to diagonal. The algorithm iteratively updates $adm(m,n)$, and stores the pointer to previous point in $\psi(m,n)$, as shown in Figure 2.58 for a case with type I constraints. Optimal alignment path is decoded from ψ by following backward indices.

Performance recording is usually analysed using STFT with 4096-sample Hamming window, and a typical hop size used is 256 samples, or 5.8 ms with 44.1 kHz sampling frequency. Therefore, DTW algorithm has to process a large amount of data – Schwarz estimates, that a three minutes of performance produce approximately 36000 frames, amounting to $1.3 \cdot 10^9$ *ldm* matrix elements to compute [502]. However, the computation time and resource cost may be reduced through improvements such as implementation of *shortcut path*, or *path pruning*.

For the purpose of score alignment it is not necessary to process evolution within a note. It is sufficient to keep only the first and the last score frame per note. Paths reduced to these frames only are referred to as shortcut paths, and their implementation reduces path matrix by 98% [416]. The other improvement, path pruning, involves keeping only the best paths in every iteration, and removing these with $adm(m,n)$ above a threshold value θ_P [502]

$$\theta_P(m) = 1.1 \min\left(adm(m-1)\right) \tag{2.80}$$

Pruning is not applied to paths between 400-frame wide corridor of selected paths and the diagonal.

There is a possibility to improve precision of DTW by implementing additional transient or onset detection algorithms [476, 473, 227]. Percussion may also be handled by combining score alignment with beat alignment [216].

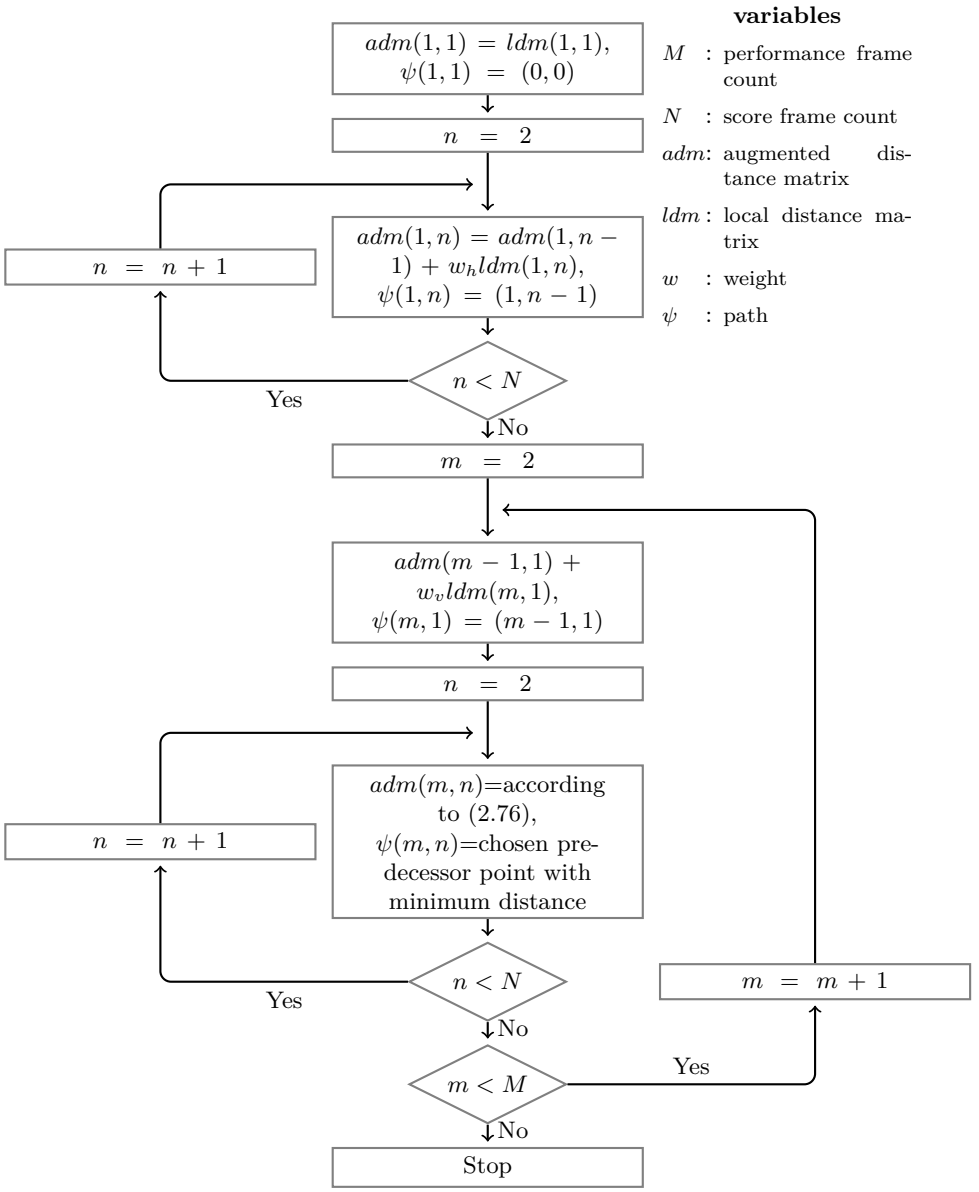


Figure 2.58. Application of Viterbi algorithm to search for optimal path ψ in score aligning procedure
Source: author's elaboration, based on Schwarz [502]

Note-level units are submitted to further sub-segmentation to separate units containing attack, sustain, and release phases [228]. A crude solution is to blindly define the first 100 ms of a note as attack, and the last 100 ms as release. Not all instruments and not all performance techniques produce sustain phase, therefore attack may immediately continue into release. An attack phase with a preceding release phase are considered together a transition segment. Additionally a *dinote* units are produced. They consist of two *seminotes*, and their transition segment. Seminotes are split in the middle of the note – usually in sustain section.

2.2.4.2. Analysis and Descriptors

Characteristics of sounds used in CS are stored within descriptors (Tab. 2.9). Three types of descriptors are utilised [501]. *Static descriptors* are constant over a whole unit. *Dynamic descriptors* change over a unit duration and are usually the result of unit analysis. They are stored in a reduced form, as a vector of characteristic values (Tab. 2.10). The last type, *category descriptors* assign a unit to a particular category and subcategories.

Table 2.9. Classes and examples of descriptors used in CS, according to Schwarz [501]

Class	Examples of descriptors
Unit	Start time, end time, duration, type
Source	Class and subclasses of the source sound, performance style
Score	MIDI note number, polyphony, lyrics, other information included in score, arbitrary subjective information
Signal	Energy, fundamental frequency, zero crossing rate, harmonics cutoff frequency
Perceptual	Loudness, sharpness, timbral width
Spectral	Spectral centroid, spectral tilt, spectral spread, spectral asymmetry
Harmonic	Harmonic energy ratio, harmonic parity, tristimulus, harmonic deviation

In cases where descriptor data is not assigned to units on the basis of score, or arbitrarily, it is obtained through data analysis which may be performed by external programs. Signal, perceptual, spectral, and harmonics descriptors (Tab. 2.9) are calculated using standard signal processing or psychoacoustic methods [321]. A detailed information on descriptors and characteristic values used in Caterpillar system may be found in works of Schwarz [500, 501, 502].

2.2.4.3. Target

Target is specified in a form of symbolic or audio score [500]. Symbolic score may be provided as a MIDI file, or in another digital representation, such as MusicXML [213] or Lilypond [406]. Schwarz discusses the requirements for symbolic score format adequate for CS and describes a number of possibilities [502]. Some of these representations are aimed towards performance and control over sound synthesis, while others tend more towards score publishing. Therefore they slightly differ in

types of stored information, although it is generally possible to perform a conversion between them. The most basic data in a symbolic score are note parameters such as pitches and durations. Score may also contain performance data, i.e. articulation, dynamics, or changes of tempo. Additionally, vocal parts contain lyrics.

The second form of target requires a different approach. A recording to be resynthesized is referred to as the audio score, and in order to provide target information it needs to be analysed in a way similar to source sounds in the segmentation process. Some descriptors (Tab. 2.9 and 2.10) need to be assigned as well.

Target score is segmented into a sequence of target units t_τ, each with a set of features that needs to be generated. Segmentation of symbolic score divides it into separate notes with all note data obtained from the score. In case of audio score a recording is segmented with the same tools as database recordings. In both cases the result of score segmentation undergoes further segmentation into attack, sustain, and release phases. It provides more combinations for the selection algorithm in following stages.

Table 2.10. Characteristic values of dynamic descriptors, as implemented in Caterpillar CS system [501]

Characteristic values	Comment
Arithmetic mean, geometric mean, standard deviation, minimum and maximum value, range	—
Slope, curvature, approximation residual	—
Start and end values, curve slopes at start and end of unit	Needed to calculate continuity of concatenation
Envelope: direct and inverse AR, i.e. attack and release time relative to extremal values, ADSR	—
Temporal centre of gravity	Location of most important elevation or depression in a descriptor curve
Normalised, 5-band Fourier spectrum of the descriptor, and 4 first spectral moments	Reveals slow or rapid descriptor change and possible oscillations

2.2.4.4. Database

Data stored within a database includes references to sound files, units, and their descriptors. Depending on the complexity of CS system and expected amount of data, various database systems may be utilised. In simpler cases flat analysis data files and SDIF (Sound Description Interchange Format) [90] unit feature files may be sufficient [500], but higher quality synthesis requires significantly larger databases, where it is advantageous to use a relational database management system (DBMS) [501], even at a cost of performance loss. In relational DBMS data is accessed using a declarative query language SQL, and since it does not depend on physical data organisation, such database is flexible, scalable, and open for future changes. Further advantages

are data consistency, and client-server architecture with possible networked operation. Therefore one, central database may safely serve many distributed synthesis systems. An example of CS database schema used in Caterpillar system [501] is presented in Figure 2.59.

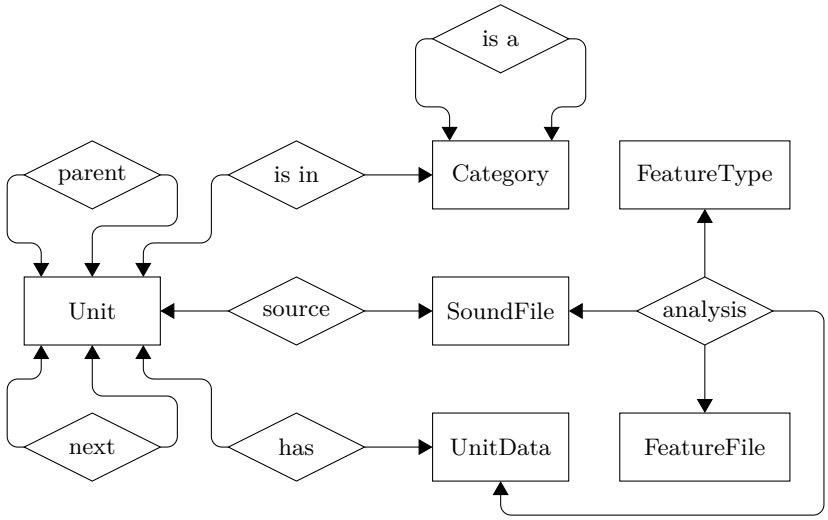


Figure 2.59. An overview of a database schema used in Caterpillar CS system; rectangles depict entities, diamonds depict relationships
Source: author's elaboration, based on Schwarz [501, 502]

2.2.4.5. Selection

There are multiple ways to perform selection of database units u_i that best match target units t_τ. The first approach utilised in early implementations of Caterpillar system developed by Schwarz [501] is based on two cost functions. Similarity between u_i and t_τ is estimated by a *target cost* that also considers r units around the target. Quality of concatenation between u_i and preceding $u\prime$ is estimated by a *concatenation cost*.

Target cost C^t may be expressed as [501]

$$C^t(u_i, t_\tau) = \sum_{k=1}^{p} w_k^t C_k^t(u_i, t_\tau) \tag{2.81}$$

where C_k^t is the individual feature distance function, w_k^t is the feature weight, and p is the number of features. In order to prefer units that belong to the same context, a *context cost* is estimated [501]

$$C^x(u_i, t_\tau) = \sum_{j=-r}^{r} w_j^x C^t(u_{i+j}, t_{\tau+j}) \tag{2.82}$$

where r is the maximal considered unit distance, and weights w_j decrease with distance j. Depending on the descriptor, different distance functions or lookup tables may need to be used.

Discontinuity resulting from the concatenation of units u_i and u_j is expressed by concatenation cost [501]

$$C^c(u_i, u_j) = \sum_{k=1}^{q} w_k^c C_k^c(u_i, u_j) \tag{2.83}$$

where C_k^t is the feature concatenation cost function, w_k^c is its weight, and q is the number of concatenated features estimated. Cost may vary depending on a type of unit.

Assuming that database contains N units, target sequence has T units, $w^t C^x$ is the state occupancy cost b_{ij} and $w^c C^c$ is the transition cost a_{ij}, the Viterbi algorithm [596] is used to find an optimal path ψ (Fig. 2.60) and sequence of units s_τ (Fig. 2.61). The sequence is found by decoding ψ in reverse order, beginning with the endpoint k with the least global cost a_{kT}. Performance of the algorithm may be improved by pruning the state transition network, in a manner similar to path pruning in score alignment.

More flexible approach to selection involves reformulating the problem as a constraint satisfaction problem (CSP) using the *adaptive local search* (ALS) algorithm [564, 121]. Constraint satisfaction error function may be expressed by target and concatenation cost.

Sequence of units indices to select is stored within variables V_i. A total cost of selection is expressed by the *global cost function* that is to be minimised [501]

$$C^s = w^t C^x(u_{V_1}, t_1) + \sum_{i=2}^{T} \left(w^c C^c(u_{V_{i-1}}, u_{V_i}) + w^t C^x(u_{V_i}, t_i) \right) \tag{2.84}$$

where each unit constraint $\boldsymbol{C}_i$ comprises V_{i-1} through V_{i+1}. The error function for one constraint takes the following form [501]

$$E(\boldsymbol{C}_i) = w^t C^x(u_{V_i}, t_i) + w^c C^c(u_{V_{i-1}}, u_{V_i}) + w^c C^c(u_{V_i}, u_{V_{i+1}}) \tag{2.85}$$

consisting of target cost of unit u_{V_i} and concatenation cost of its two adjacent units.

AVL starts with a random configuration of variables, and repeats the following four steps in a loop [501]:

1) compute error for each constraint, distribute it over the constraint variables; V_i is responsible for a target error $w^t C^x(u_{V_i}, t_i)$, as well as for half of both concatenation errors $w^c C^c(u_{V_{i-1}}, u_{V_i})$, and $w^c C^c(u_{V_i}, u_{V_{i+1}})$;

2) replace a unit from the variable that has the largest error and is not marked as taboo with the best matching unit u_j, where

$$j = \arg \min_{1 \leq j \leq N} (E(\boldsymbol{C}_i)) \tag{2.86}$$

3) mark the variable as taboo for a given number of iterations if there is no unit with lower error;

4) restart with a new random configuration of variables if all variables were marked as taboo.

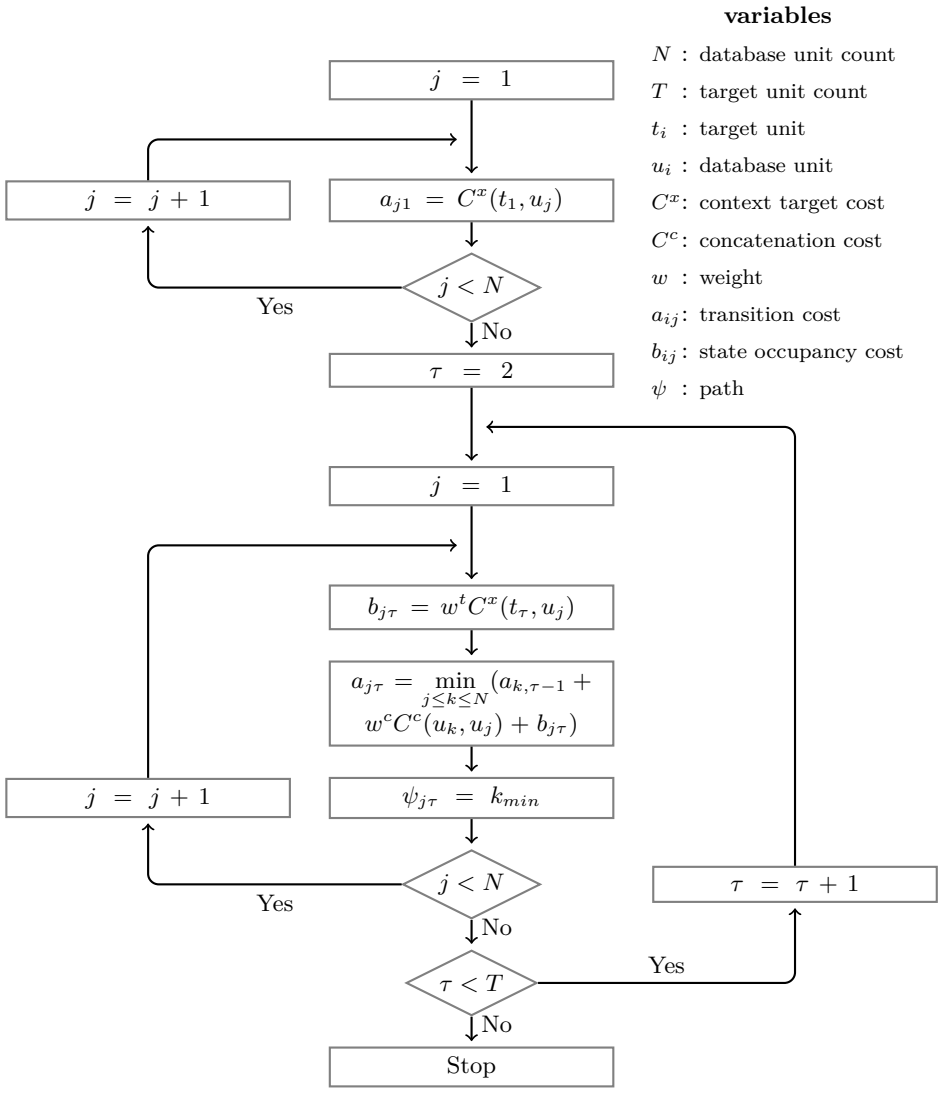


Figure 2.60. Application of Viterbi algorithm to search for the optimal path ψ in unit selection process
Source: author's elaboration, based on Schwarz [501]

The loop breaks and the search is finished if either the global cost is lower then a given threshold, or a previously assumed maximum number of iterations is reached. There are four additional constraints: *all different*, *unit ban*, *unit forcing*, and *unit lock*. The first two may be enforced using high penalty values: a second appearance of a unit within a selection may be prohibited, and particular units may be banned. In reverse, units may be forced by lowering error for variables they are contained within. Finally, selected units may be locked by setting variables that contain them as taboo.

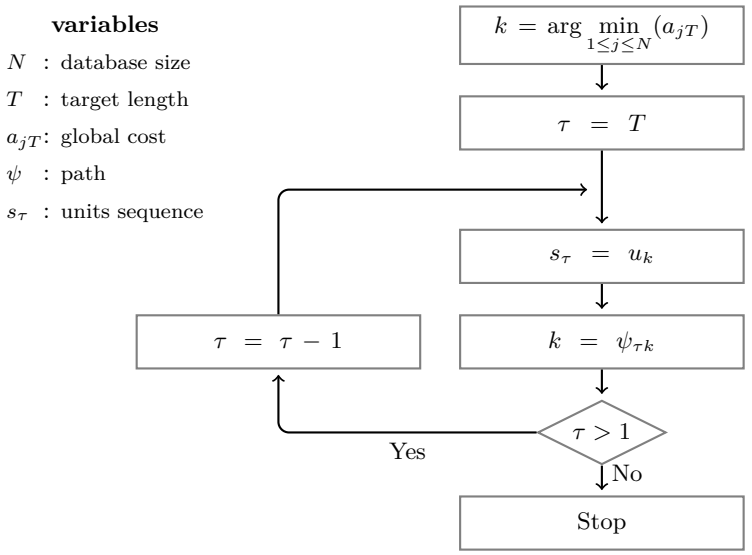


Figure 2.61. Decoding path ψ to find the optimal units sequence s_τ in a unit selection process
Source: author's elaboration, based on Schwarz [501]

Out of two presented approaches, the linear path-search unit selection is more efficient, and may be utilised in real time. Application of CSP unit selection makes the process less efficient, but more flexible, allowing a better control over selected units through aforementioned additional constraints.

As an alternative selection method Ó Nuanáin et al. propose to use k-best hidden Markov model decoding [411]. Such approach allows to solve an issue of HMMs when paired with Viterbi decoding, i.e. producing only the highest probable state sequence, while solutions preferred in musical applications should allow alternatives to be evaluated and chosen by the user.

2.2.4.6. Synthesis

The final step – synthesis – differs depending on application. If synthesis target is a musical score, it determines positions of units on a time-line. Otherwise, e.g. in speech synthesis, unit durations determine their placement. The process itself is performed in two stages, transformation and concatenation.

Transformation (Tab. 2.11) may need to be applied to reduce target distance and concatenation distance. The former makes database units better match target, while the latter matches edges of adjacent units to reduce transition artifacts. The only method to change duration of sound samples without degrading sound quality is shortening. Alternatively, if there is an option to switch from recorded samples in a database to a different sound representation, then e.g. pitch synchronous overlap add (PSOLA) [567] or additive synthesis provide more duration-manipulation possibilities. A number and complexity of transformations required may be reduced by preparing larger databases. If units were appropriately transformed, they may be concatenated using a simple crossfade with overlapping.

Table 2.11. Unit transformations applied in the synthesis stage of concatenative synthesis, according to Schwarz [502]

Parameter changed	**Method of transformation**
Fundamental frequency	Resampling
Energy	Multiplication
Spectral envelope	Filtering
Duration	Shortening, time-granulation

2.2.4.7. High Level Instrument Synthesis

The most prominent use case of CS is reproducing instrument parts from a score, using high-level control, and leaving fine details to be filled in by the database units. Schwarz refers to such scenario as a *high level instrument synthesis*.

CS is able to reproduce details that are a result of various performance techniques or phrasing, usually manifesting themselves as subtle irregularities in characteristics such as energy, pitch, duration, or timbre. There are two cases where synthesis methods other then CS may also produce such performance details. The first one is a live performance of a musician playing a synthesizer. In this case the details are produced by the musician on a skill, knowledge, and experience basis, provided that the synthesizer allows to control fine details of produced sound, and is supplemented with appropriately sophisticated controller to encode all control gestures into synthesis parameters.

The other possibility to introduce fine details into a synthetic performance is to pair a synthesizer with an implementation of performance rules that will analyse the score and modify synthesis parameters accordingly, beyond coarse values provided in score. Rules may be constituted in a number of ways [158], such as *analysis by synthesis* [84, 195, 338] or *analysis by measurement* [199] approaches, using learning systems such as artificial neural networks [603, 82, 604, 605], fuzzy logic [83], or various other models [375, 296], including measurement-based complex modelling of virtual performers [429].

Rule-based systems may produce results better than direct reproduction of score data, but they still struggle to mimic human performance. On the other hand, a CS system is aware of context of units it uses, and is able to produce natural transitions

without imposing additional performance rules, on a score basis only. Performance-induced irregularities are present in database units, produced from recordings of human performances, hence if a proper path has been found in selection stage, they are transferred to the CS output signal.

High level instrument synthesis with CS method implemented in Caterpillar system [502] utilises dinote segments as well as sub-segments containing attack, sustain, and release phases. Since both, segmentation, and pitch analysis may produce errors, it is necessary to perform sanity checks to verify dinote units pitch curves, e.g. on a basis of pitch transition width and pitch range. During selection the highest weight is given to pitch descriptor, since the melody has to be matched, and lower to unit type descriptor. The latter forces seminote units at both ends of synthesized phrase, and dinotes in the middle. Additionally, a very low weight is given to duration descriptor to prefer units longer then a threshold value.

2.2.4.8. Real-Time Concatenative Synthesis

High complexity with multi-stage analysis process, and control over the synthesizer through a musical score or recording of a musical piece, common for early implementations, might imply that CS is not a method suitable for real-time performance. There is, however, a number of real-time corpus-based CS implementations. According to Schwarz [505] they may be grouped in two classes. In the first one matching process is based on descriptors, in the second one – on spectrum. Examples of descriptor-based approach include synthesis systems such as CataRT [504], MoSievius [324], Synful [336], or Ringomatic [23]. SoundSpotter system [104] is an example of spectral-based approach. Systems based on spectral matching involve control on a signal processing level, while descriptor-based systems allow to control synthesis process on a score level, which is better suited to musical applications.

CataRT [504, 505] is an implementation of real-time CS that is corpus-based and uses descriptors, thus adhering to content-based processing paradigm. Descriptors form a multi-dimensional space filled with sound units. The user is given control over a target point and selection radius within a lower-dimensional projection of the descriptors space. Selection algorithm chooses units that are closest to a target and are placed inside a selection radius. Selection and triggering of units are independent. Due to real-time constraints concatenation is carried out in a simplified manner. The only transformations are adjustments of pitch and loudness, as well as a short crossfade. Quality of concatenation is not considered.

Diagram of CataRT is presented in Figure 2.56. The corpus is fed with audio input which may be stored in a database. During synthesis data is retrieved from the corpus. Different forms of audio input data, as well as various possibilities of its analysis are presented in Figure 2.62. In the simplest case input data is provided as an analysed audio file, with markers and descriptors already prepared. It can be added to the corpus directly, without additional processing. If descriptors are missing, they need to be prepared, and analysis is required. In case of audio data without descriptors and markers, additional onset detection may be required, depending on contents of the recording. The analysis required in this stage takes place in real time, therefore the data, instead from a file, may as well originate from a real-time

input. Another common practice is to constantly record a musician performing on stage, analyse the data in real time, and use last several minutes as a corpus, which is constantly updated. In such case the synthesizer is controlled by another performer using a computer and controlling selection of units, as presented in Figure 2.63.

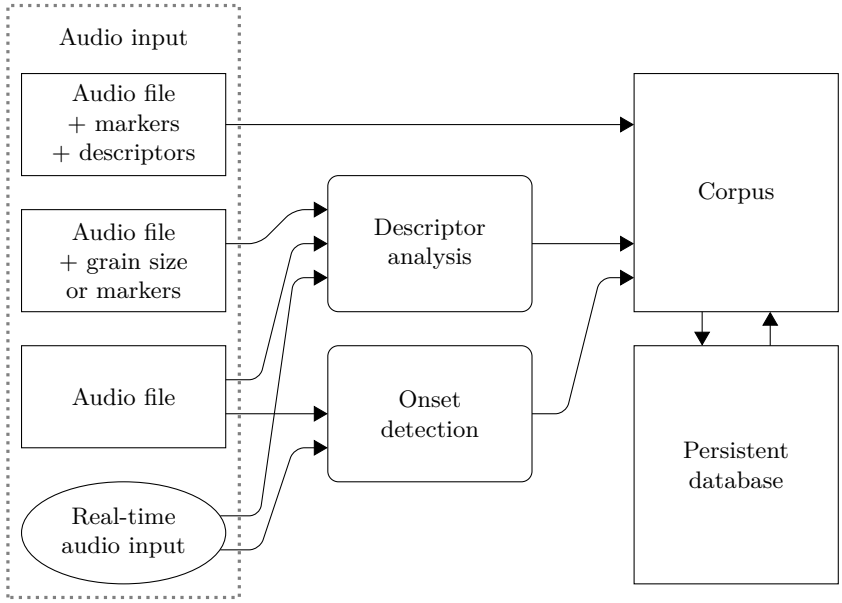


Figure 2.62. Analysis process in real-time corpus-based concatenative synthesis
Source: author's elaboration, based on Schwarz et al. [505]

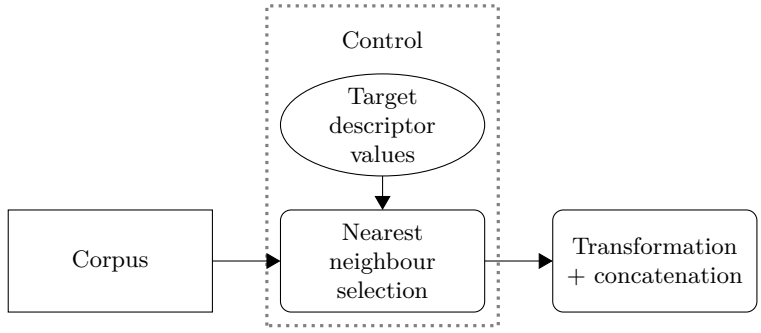


Figure 2.63. Synthesis process in real-time corpus-based concatenative synthesis system
Source: author's elaboration, based on Schwarz et al. [505]

Implementation of CS for real-time performance requires a number of simplifications [505]. If source audio needs to be segmented into units, viable options include arbitrary grain segmentation, splitting on silence sections with a given threshold level,

or using the *YIN* algorithm [153, 152]. Descriptors analysis may be based on MPEG-7 signal, perceptual, spectral, and harmonic descriptors [426, 543], while the YIN algorithm can provide values of f_0, aperiodicity, and loudness. For the sake of efficiency, database is simplified, and based on Sound Description Interchange Format (SDIF) [90].

The crucial difference is the simplification of selection stage. In interactive synthesis it is not possible to search for globally optimal units path. Selection is carried out among units that are close to the current position in a descriptor space. The distance is calculated using squared Mahalanobis distance [345]

$$d = \frac{(x-\mu)^2}{\sigma} \tag{2.87}$$

where x is the position in a descriptor space, μ is the matrix of unit data, and σ is the standard deviation of each descriptor over the corpus. For a specified selection radius r a unit with $d < r^2$ is selected. It may be the closest unit, one of k closest units, or a randomly chosen unit. During synthesis, it is possible to precisely specify desired pitch and loudness values. The original values are stored in the unit descriptor, and may be altered using resampling and multiplication.

Comparing to methods such as subtractive or sampling synthesis, real time CS is controlled in a different way, involving computer program with a graphical user interface. In case of CataRT [504, 505] the user interface presents two-dimensional descriptor space with points symbolising units, and a user-controllable circle symbolising selection range. Playback of new units may be triggered in various ways, according to selected *modes* presented in Table 2.12.

Table 2.12. Unit triggering modes in real-time concatenative synthesis, according to Schwarz [505]

Mode	**Unit triggering principle**
Bow	Unit is triggered each time a mouse is moved
Fence	When a different unit becomes the closest one it is triggered
Beat	Units are triggered in regular time intervals
Chain	New unit is triggered when the previous one ends
Continue	Unit that followed the last one in the original recording is triggered
Seq	Units are triggered by outside source, such as a sequencer

Further development in the area of real-time CS includes the EarGram system proposed by Bernardes et al. [56], intended to explore large database of audio snippets. EarGram is an interactive tool for improvisation and composition. It is based on multiple views of the database, including an interactive scatter plot. A number of unit triggering strategies is available.

2.2.4.9. Expressive Concatenative Synthesis

Schwarz formulation of the concatenative synthesis method [502] gives flexible definitions that encompass a broad class of techniques with various aims and appli-

cations. However, it may seem to lack some implementation details. The particulars are left to fill in for the actual techniques aimed at more focused applications.

Maestre et al. [343, 344] proposed a CS variant, referred to as expressive concatenative synthesis, that emphasizes expressive component of musical performance. While usually synthesizers require manual adjustments in order to produce instrument performances that may be considered expressive on a score basis only, expressive CS is aimed at producing such output automatically by utilising a set of expressive performance recordings and *performance models* trained using inductive logic-programming techniques. Thus it supplements the set of CS tools with more advanced mechanisms dealing with unit selection and transformations.

Introductory research on analysis and concatenative synthesis of expressive performance was presented by Maestre and Gómez [342], as well as Ramirez et al. [457, 458]. Expressive CS performs audio data segmentation and characterisation based on fundamental frequency and energy. Segmentation is carried out at different temporal levels, i.e. note, intra-note, and note-to-note transition. Units are processed using granular time-stretching, pitch-shifting, and energy transformation. During synthesis stage a phase-vocoder is applied to concatenate samples corresponding to entire performed notes of arbitrary durations.

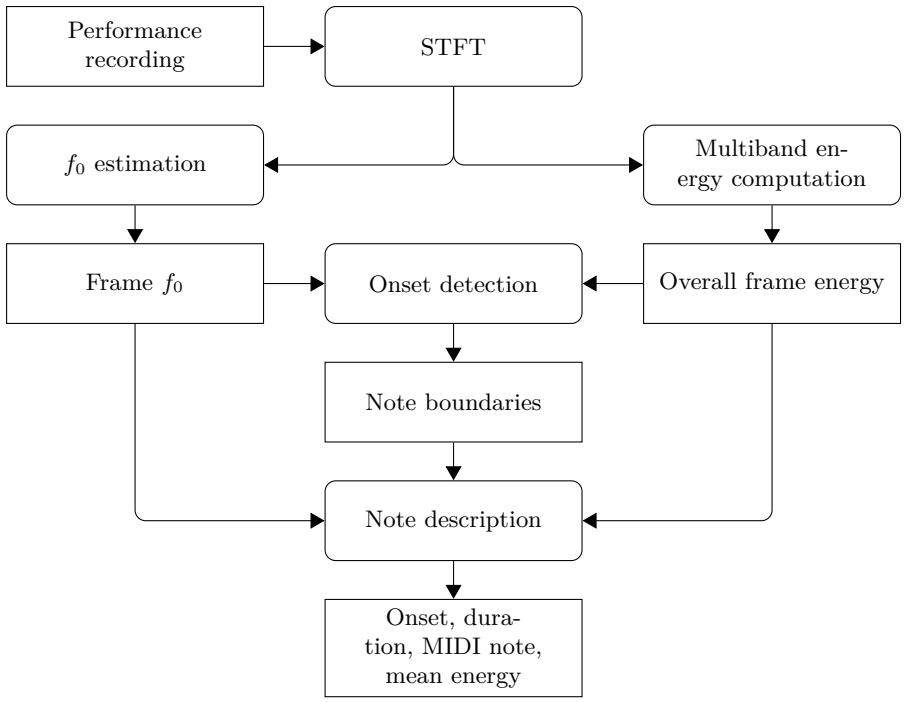


Figure 2.64. Melodic description process in expressive CS
Source: author's elaboration, based on Maestre et al. [344]

Analysis of expressive performance recording begins with STFT and computation of low-level descriptors for each frame. Next, melodic description is carried out to determine MIDI pitch, onset, and duration of notes (Fig. 2.64). On the basis of note descriptors a musical analysis is performed, according to Narmour's theory of perception and cognition of melodies [401, 402], which assumes the existence of *implicative intervals*. Such interval implies its successor according to two principles, i.e. *registral direction* and *intervallic difference*. Interestingly, both are in accordance with strict counterpoint rules concerning construction of a proper melodic line [489].

Apart from a high-level musical analysis, energy envelopes are studied to determine internal features of segmented notes and their transitions, i.e. attack, sustain, and release sub-segments. Releases with following attacks are considered transition segments. Their energy and f_0 contours are utilised to determine articulation-related descriptors, such as the legato descriptor [344]

$$LEG = \frac{A_1}{A_1 + A_2} = \frac{\int_{t_{init}}^{t_{end}} \left(L_t(t) - E_{XX}(t)\right) dt}{\int_{t_{init}}^{t_{end}} L_t(t) dt} \tag{2.88}$$

where A_1 is the area between the schematic energy contour and the line L_t joining start and end points of the transition segment, A_2 is the area below the energy contour, and E_{XX} is the energy envelope, as shown in Figure 2.65. $LEG \in [0, 1]$ assumes smaller values in smoother transitions, and larger – in more detached.

Notes stored in a database are classified according to their context in the original performance, and to their internal structure [344]. Context is considered as a type of previous and next segment, which can be either silence (SIL) or another note. Therefore a note n can belong to one of four articulation classes, i.e. SIL-n-SIL, SIL-n-$NOTE$, $NOTE$-n-SIL, or $NOTE$-n-$NOTE$, and it is considered a strict constraint during sample selection. Notes within each group are divided into several clusters by applying k-means clustering to a set of intra-note features: attack level, sustain duration, sustain slope, spectral centroid, and spectral tilt. In $NOTE$-n-SIL group $k = 3$, otherwise $k = 2$ is sufficient. Additionally, cases with more or less than one score note represented by a single database unit can be considered, e.g. consolidation, fragmentation, or ornamentation. Table 2.13 contains a complete list of descriptors used in expressive CS, as implemented by Maestre et al. [344].

In order to synthesize an expressive audio a performance model is applied to predict expressive-related modifications in relation to the strict score reproduction [457], as shown in Table 2.14. The model is obtained by applying inductive logic programming techniques in a form of TILDE inductive algorithm [69] – a first-order logic extension of the C4.5 decision-tree algorithm.

Predictions of the performance model and database samples are utilised to generate output audio sequence in a process depicted in Figure 2.66. Samples are analysed in spectral domain and phase vocoder is applied for time, amplitude, and frequency transformations. The actual concatenation stage is performed using generic CS approach with transformation and selection cost calculated to determine the most suitable path.

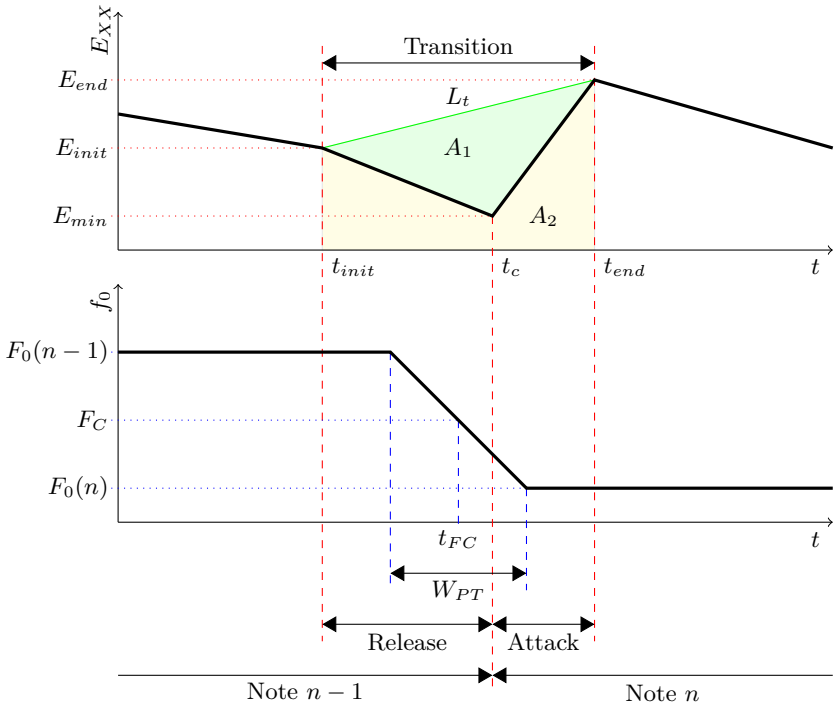


Figure 2.65. Schematic view of the transition segment; E_{XX} is the energy envelope, E_{init} is the value of energy envelope at the start (t_{init}) of the transition segment, E_{min} is the minimal energy in the transition segment at t_c, E_{end} is the energy at the end (t_{end}) of the transition segment, L_t is the line connecting energy envelope at t_{init} and t_{end}, A_1 is the area between the envelope and L_1, A_2 is the area below the envelope, $F_0(n-1)$ and $F_0(n)$ are f_0 values of the note $n-1$ and n, W_{PT} is the width of a pitch transition, t_{FC} is its midpoint, and F_C is the f_0 value in the midpoint; values required to represent actual envelopes with linear segments are calculated during automatic segmentation
Source: author's elaboration, based on Maestre et al. [344]

Transformation cost is a weighted sum of costs resulting from three kinds of transformation [344]

$$C_T = \sum_{i=1}^{N_s} w_D C_D(i) + w_F C_F(i) + w_E C_E(i) \tag{2.89}$$

where N_s is the number of note samples within a path, w are the weights, C_D is the duration transformation cost, C_F is the frequency transformation cost, and C_E is the energy transformation cost. Duration transformation cost is calculated according to the following formula [344]

$$C_D = \frac{\sum_{i=1}^{N_s} |\log_2(S_T(i))|^2}{\sum_{i=1}^{N_s} |\log_2(S_T(i))|} \tag{2.90}$$

where S_T is the time-stretch factor. Frequency transformation cost is calculated as [344]

$$C_F = \left| \log_2 \left(\frac{F_{0,Pred}}{F_{0,DB}} \right) \right| \tag{2.91}$$

where $F_{0,Pred}$ and $F_{0,DB}$ are fundamental frequency predicted and retrieved from a database, respectively. Finally, cost of energy transformation is expressed as [344]

$$C_E = \frac{1}{2} \left| \log_2 \left(\frac{E_{Pred}}{E_{DB}} \right) \right| \tag{2.92}$$

where E_{Pred} and E_{DB} are energy predicted and retrieved from a database, respectively. Complete contents of a database are given in Table 2.13, and predictions – in Table 2.14.

Table 2.13. Descriptors used in expressive CS, according to Maestre et al. [344]

Group	Descriptors
Melody and dynamics	Pitch, onset time, duration, alteration, mean energy
Context	Metrical strength, Narmour group, articulation group, duration of previous and next, pitch of previous and next
Timbre	Mean spectral centroid, mean spectral tilt
Intra-note	Attack level, sustain duration, sustain slope, legato to previous, legato to next, note change time, transition init and end time, pitch step width and centre time
Classification	Cluster number

Table 2.14. Performance model predictions in expressive CS, according to Maestre et al. [344]

Level	Transformations	Affected
Note	Duration transformation	Duration
	Onset deviation	Onset
	Energy variation	Energy
	Note alterations (ornaments)	Alteration
Transition	Type of transition (*legato*/*staccato*)	*LEG* (2.88)
Intra-note	Notes divided into a set of clusters based on intra-note features; given a musical context of a note trained classifier predicts a cluster	Attack level
		Sustain duration
		Sustain slope
		Spectral centroid
		Spectral tilt

Like transformation cost C_T, the concatenation cost C_C is also a weighted sum [344]

$$C_C = \sum_{i=1}^{N_C} w_L C_L(i) + w_I C_I(i) + w_P C_P(i) \tag{2.93}$$

where N_C is the number of concatenations, w are the weights, C_L is the legato cost, C_I is the interval cost, and C_P is the continuity cost. The following formula expresses the legato cost [344]

$$C_L = |LEG_{Pred} - LEG_{DBrec}| \tag{2.94}$$

where LEG_{Pred} is the predicted value of legato descriptor, and LEG_{DBrec} is the pre-computed legato value of two samples considered for concatenation. Interval cost is expressed as [344]

$$C_I = \left| \frac{|I_{Pred} - I_{Lright}| + |I_{Pred} - I_{Rleft}|}{I_{Pred}} \right| \tag{2.95}$$

where I_{Pred} is the target interval, I_{Rleft} is the interval between the candidate for the sample following in the transition and its predecessor in the database, and I_{Lright} is the candidate for the sample preceding in the transition and its successor in the database. Continuity cost C_P favours usage of consecutive samples from the database recordings. It is either zero, in case of consecutive samples, or unity in other cases.

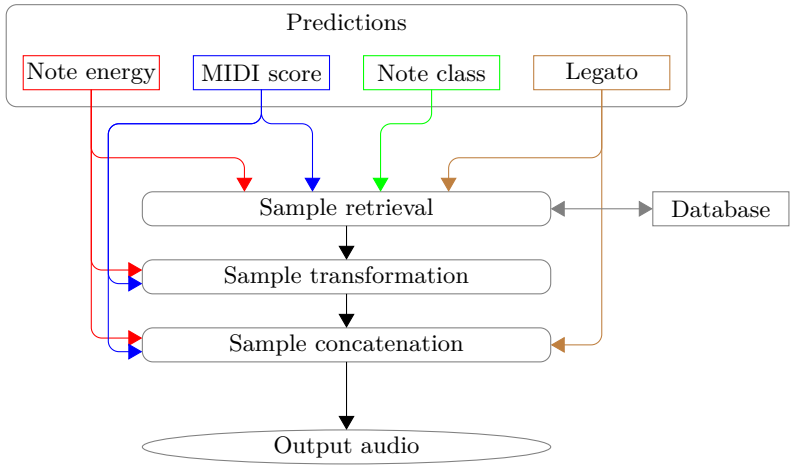


Figure 2.66. Synthesis stage in expressive CS
Source: author's elaboration, based on Maestre et al. [344]

Before concatenation samples are transformed (Fig. 2.66) using phase vocoder techniques [14, 74, 320] to match target description. Duration transformation is applied by time stretching with variable factor, related to intra-note segment. Thus the

transformation is carried out in the sustain segment, and blurring of transitions does not occur. Spectral shape and f_0 in new frames is linearly interpolated from the surrounding frames. Concatenation discontinuities in amplitude, pitch, and timbre are corrected. Energy and pitch are smoothed using cubic splines within a region of several frames. Additionally, energy curve is corrected for the value of E_{min} (Fig. 2.65) to match the legato prediction. Timbre is smoothed by spectral-shape interpolation.

2.2.4.10. Other Variants of Concatenative Synthesis

Though the term has been later adapted to serve as an alternative name to CS, the **musical mosaicing** was initially developed by Zils and Pachet as a mechanism to automatically generate a sequence of sound samples. The sequence is specified through high-level properties only. User-provided properties are interpreted and utilised as a basis for constraint solving problem [619]. Typically CS is aimed at producing continuous sequences. However, since musical mosaicing was primarily aimed at assisting composition work in sample-based music genres, the algorithm allows to define various constraint classes, such as difference, cardinality, or distribution, to produce sequences that differ in expressive qualities.

Segments of the sequence are referred to as variables, and are assigned with descriptors such as mean pitch, loudness, percussivity, related to variations of amplitude, and global timbre, based on spectral distribution. The method minimises a global cost function, consisting of weighted cost functions associated with particular constraints. Segment constraints set target values for a single descriptor. Sequence constraints apply to distribution, continuity, or cardinality of the whole sequence or any set of segments. They are not limited by a number of descriptors. Distribution constraints control sample placement, continuity constrains control continuity with regards to selected feature, and finally, cardinality constraints control number of samples, as well as their uniformity. The output is produced by finding a sequence satisfying the constraints. In large databases of samples a complete search would be too slow, therefore the adaptive search algorithm [121] solves the constraint system locally (Fig. 2.67). In the end, sequence quality is further refined using global transformations to correct sample transitions.

Mosaicing may be implemented as an interactive, real-time synthesis method. It was investigated and incorporated into MoSievius system by Lazier and Cook [324]. MoSievius (Fig. 2.68) is a framework that allows to conveniently implement various interactive mosaicing techniques. Real-time control over mosaicing involves manipulating source and unit selection process. As long as either of these is controlled, mosaicing is considered interactive, even if the target has been fully specified.

Later modifications to mosaicing proposed by Coleman et al. include augmenting the method with descriptor-driven transformations and adding dynamic smoothing [125, 124, 123]. In the former, chroma, mel-spaced filter banks, and energy are utilised as target descriptors modelled in order to adapt produced output to musical contexts. In the latter several source atoms are mixed using sparse signal representation techniques. In a standard path search approach applied in CS a single weighted source frame is chosen for each target frame.

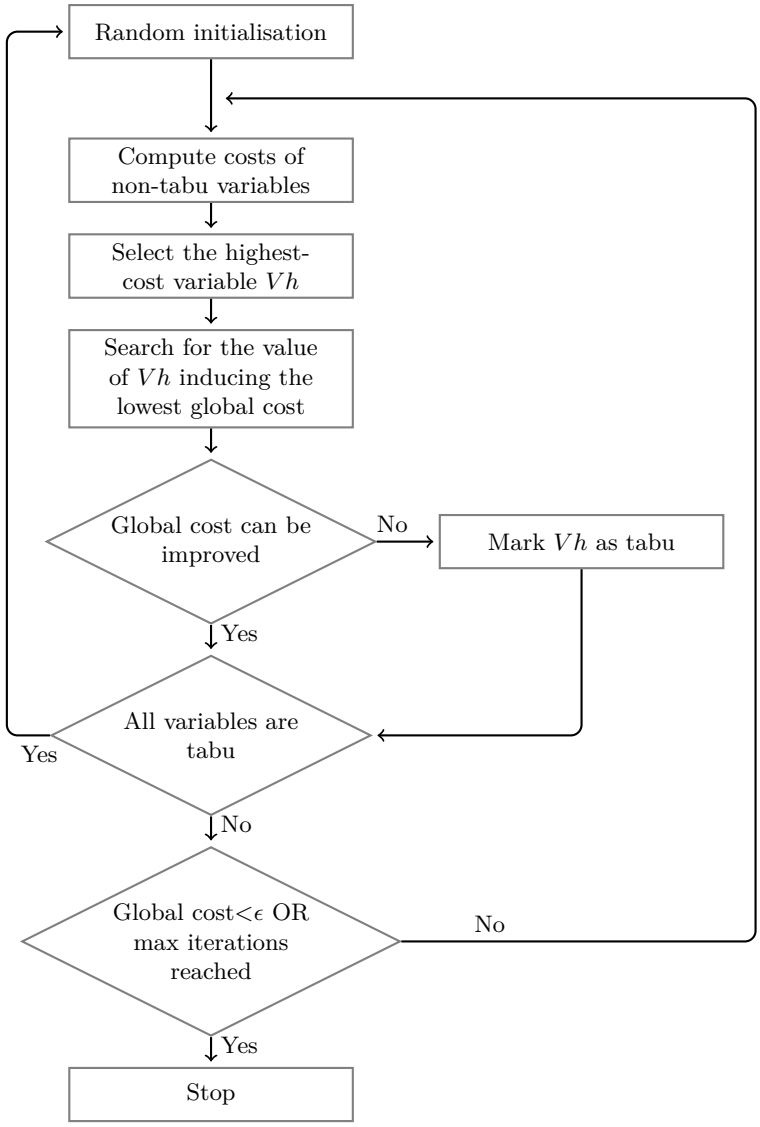


Figure 2.67. Adaptive search algorithm, as applied in musical mosaicing to find a sequence of database samples
Source: author's elaboration, based on Zils and Pachet [619]

The optimisation problem is solved by fitting the following form [123]

$$\arg\min_{s[t],w_t} \sum_{t=1}^{T} \|y_t - w_t \boldsymbol{a}_{s[t]}\|^2 + f(s[1],...,s[t]) \tag{2.96}$$

where $s[t],\ t = 1 \ldots T$ are the indices of a source frame sequence chosen for each target frame, w_t is the gain of the target frame t, and f is some function of $s[t]$. By introducing frame mixing, a single target frame can be approximated with a weighted sum of all source frames. In consequence the optimisation problem changes as follows [123]

$$\arg\min_{\boldsymbol{w}_t} \sum_{t=1}^{T} \|y_t - A\boldsymbol{w}_t\|^2 + f(\boldsymbol{w}_1, ..., \boldsymbol{w}_T) \tag{2.97}$$

where $\boldsymbol{w}_t,\ t = 1 \ldots T$ is the sequence of weight vectors, and A is the matrix of source descriptor vectors. Mixing can solve a problem of insufficiently populated databases by allowing to combine simpler features, which are usually easier to find, e.g. to produce chords. As Coleman points out [123], since (2.96) is a special case of (2.97), the distance to target in (2.97) will always be less then or equal to the case of (2.96).

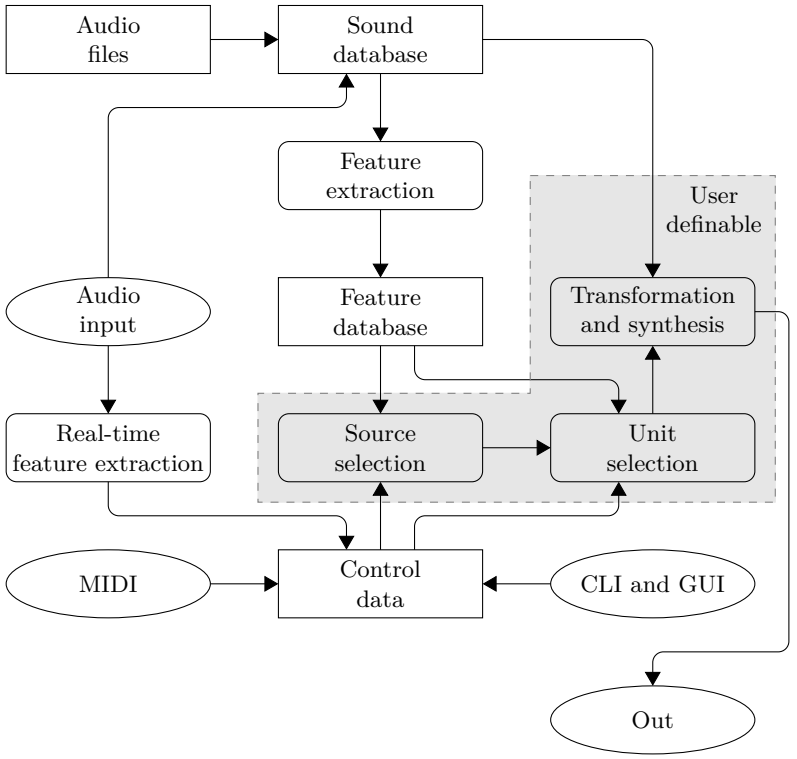


Figure 2.68. MoSievius, a framework for the implementation of various interactive mosaicing techniques
Source: author's elaboration, based on Lazier and Cook [324]

Actually, the first system to use frame mixing was proposed by Hoffman et al. [239]. It utilised a probabilistic generative model, referred to as the Shift-Invariant Mixture of Multinomials (SIMM), which can be interpreted as fully Bayesian variant of Shift-Invariant Probabilistic Latent Component Analysis [518]. Although similar in its

use of mixtures, a system designed by Coleman et al. [125] uses a different sparse projection technology, referred to as a **basis pursuit** [113].

Simon et al. [513] developed a system referred to as **Audio Analogies** aimed at producing audio output B' given audio input A', and two score inputs, A and B (Fig. 2.69). Score B is the target to be synthesized as B'. Score A and audio A', which contains a recorded performance of A, is the example pair used to synthesize B in such a way, that relation of B and B' is the same as A and A'.

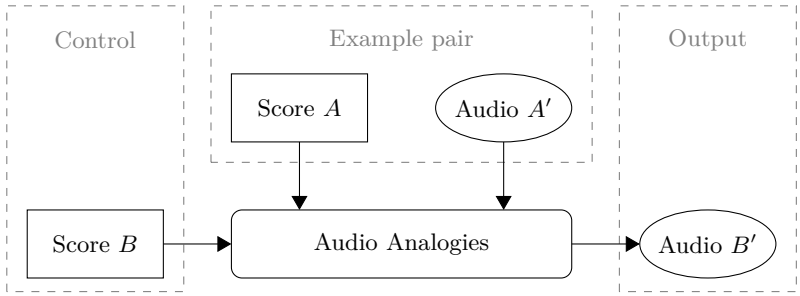


Figure 2.69. Working principle of the Audio Analogies synthesis system; relation of produced audio B' to score B is the same as in the example pair of recorded performance A' and its score A
Source: author's elaboration, based on Simon et al. [513]

The score B may be regarded as a high-level control data. Output signal B' is produced by concatenating sound units, referred to as *frames*, obtained through segmentation of A using A'. Frames contain either single notes or pairs of adjacent notes. Dynamic programming algorithm searches for the optimal sequence of frames from A', considering cost function of two objectives: match between each frame in A' and B', and coherence of the sequence with respect to A'. The optimal sequence is found by minimising the following expression [513]

$$C = \alpha \sum_{i=1}^{n} C_{match}(i, S_i) + (1 - \alpha) \sum_{i=1}^{n-1} C_{transition}(i, S_i, S_{i+1}) \tag{2.98}$$

where S is the sequence of n frames, C_{match} is the cost of matching a frame from A' to B', $C_{transition}$ is the cost of concatenation between two adjacent frames, and $\alpha \in (0, 1)$ is the parameter that controls the tradeoff between output better matching input score (α close to 1), and output being coherent with respect to A' (α close to 0). Example may contain a number of score-performance pairs, leading to better matching at the expense of compute time. Furthermore, frames from A' may require modifications to match frames in B'. In such cases pitch and duration are manipulated using resampling and synchronised overlap-add (SOLA) [480]. In concatenation stage adjacent frames that contain pairs of notes are first lined up using cross-correlation to align phases of shared note, and then blended using linear crossfade. A system similar to Audio Analogies, though polyphonic, was described by Dannenberg [151].

While concatenative or mosaicing synthesis is usually controlled using graphical interfaces, a different approach was presented by Janer and de Boer, who developed mosaicing synthesizer controlled with a voice signal [269]. Such control method is referred to as a voice-driven synthesis [268]. Voice input is segmented, and each segment is analysed to extract features (Tab. 2.15) that will be mapped to audio source features.

Table 2.15. Features extracted from segmented voice input in a voice-driven mosaicing synthesis, according to Janer and de Boer [269]

Feature	Characteristic
Audio centroid	Impulsiveness
Energy	Loudness
Spectrum flatness	Harmonicity or noisiness
Harmonic pitch class profiles	Tonal description
Mel-Frequency Cepstrum Coefficients	Timbre
Spectral Centroid	Brightness

Musical concatenative synthesis originates from speech synthesis methods. Strong traces of this origin remain in earlier CS systems aimed at synthesizing singing voice, which requires a combination of both, musical and speech synthesis. Early research on **concatenative singing voice synthesis** was presented in works of Macon et al. and Meron [341, 363]. Singing voice synthesis presented by Bonada and Loscos [74] and later by Bonada and Serra [75] utilises a database of diphoneme samples to search for the best sequence, which is assembled using phase vocoder and spectral concatenation methods. The method yields good results, and was implemented in commercial synthesizer, Yamaha Vocaloid [293].

A method proposed by Kobayashi [302], **sound clustering synthesis**, uses STFT analysis to extract spectral data containing sound transitions features. Database of sounds is pre-analysed and pre-clustered. Spectral match function is used to resynthesize a target inheriting transition features of analysed sounds (Fig. 2.70). The method conserves the association of consecutive frame clusters.

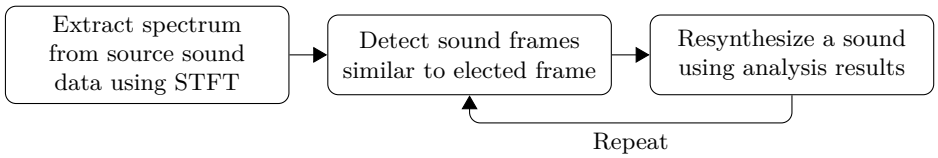


Figure 2.70. Steps of the sound clustering synthesis
Source: author's elaboration, based on Kobayashi [302]

Reconstructive phrase modelling (RPM) [336] is able to produce expressive solo instrument performance controlled in real-time by MIDI. It relies on splicing fragments of phrases, and in this regards it shares a number of features with CS. However

it operates using hybrid waveform and additive synthesis with sine and noise components, hence it differs from the majority of other CS methods that are based mainly on waveform representation. Recordings of instrumental performances are segmented into units, or splices, representing attack, sustain, release, and transition, with various subtypes – one note can be built from several splices. A set of rules is used to convert real-time MIDI data into a synthesis target, matched through selecting closest splices, and taking into account local contexts spanning several notes. The distance function is based on pitch and loudness. Hybrid waveform-additive representation allows RPM to manipulate units with more flexibility than other CS methods. Selected units are stretched, pitch-shifted, time-shifted, and combined. Although RPM may be more constrained in applications than a general CS method due its fixed inventory and limited feature set, it is particularly well suited for high-level instrument synthesis [503], and has a commercial application – the Synful Orchestra.

GUIDAGE, a system presented by Cont et al. [130] introduces a new descriptor referred to as the Audio Oracle. It is a Markov model that allows to predict evolution and repetitions of other descriptors. The system, however, is aimed more at audio retrieval than at sound synthesis.

A number of CS implementations involve **cross-synthesis**, and may be considered as related to granular synthesis due to utilising frame-level units. Time-domain audio decomposition is often based on a sparse approximation method referred to as **matching pursuit** [351]. Collins and Sturm [126] proposed to decompose source and target with a Gabor dictionary. Weights of Gabor atoms in the target can be modulated, and source weights can be mixed with target weights [540, 541]. An alternative method, **non-negative matrix factorisation** [330], can also be utilised to perform separation of non-negative signals into parts [517]. Burred [89] designed a system that performs cross-synthesis by substituting source and target spectral components. Both sets are matched using a timbre similarity measure based on mel-frequency cepstral coefficients (MFCC). In a system designed by Fukayama and Goto [196] relative chord present in target signal is imposed on the source signal. System presented by Driedger et al. [171] factors the target spectrogram with the spectrogram of source. This allows to resynthesize the target from weighted combinations of source frames.

3. Indirect Methods

3.1. Abstract Methods

3.1.1. Frequency Modulation

If one requires a very distinctive, 'synthetic' sound, frequency modulation (FM) synthesis may be the method of choice. Interestingly, it is regarded so even though Chowning, who was its inventor and carried out initial research on its musical applications [116], intended to produce sounds with natural qualities.

The principle of modulation in general, and frequency modulation in particular, had been well known long before it became a sound synthesis method – its theory had been established [103, 66], and it was utilised in radio broadcasting. Actually, FM already had a musical application in a form of the *vibrato* (Fig. 2.4), though in case of human performers and acoustic instruments modulation frequency was insufficient to go beyond the simple effect. Initial experiments carried out by Chowning started with an extreme *vibrato*, where a temporal structure of the modulation is no longer perceived, and timbral effects begin to appear instead. Such technique proved to be an efficient way of controlling timbre in real time [470], and became the basis of a new synthesis method which, according to Chowning [116], allowed to generate diverse, dynamic spectra of either known, or unknown sounds.

Actual musical applications of FM synthesis include complex configurations of multiple generators producing various types of signals, yet the simplest FM instrument, still capable of producing rich spectra, requires only two sine oscillators: the *carrier* and the *modulator* that modulates carrier frequency, as shown in Figure 3.1. In this configuration the output signal frequency has two components. The first one is the constant carrier frequency f_C, and the second one is the oscillating output of the modulator. The instantaneous frequency value can be expressed as [61]

$$f(t) = f_C + A_M \cos(2\pi f_M t) \tag{3.1}$$

where f_M is the modulator frequency, and A_M is the modulator amplitude expressed in frequency units. A_M may be considered the peak frequency deviation. Thus the simple FM instrument shown in Figure 3.1 produces the following output signal [61]

$$u(t) = A_C \sin\left(2\pi \int_0^t \left(f_C + A_M \cos\left(2\pi f_M t'\right)\right) dt'\right) \tag{3.2}$$

where A_C is the carrier oscillator amplitude.

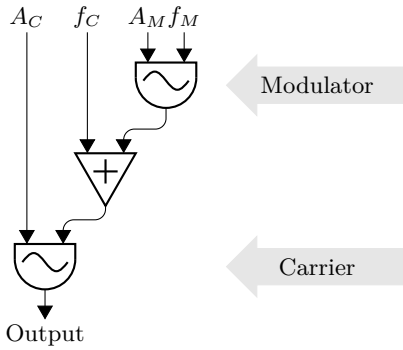


Figure 3.1. A simple FM instrument with two sine oscillators: carrier and modulator; A_C is the carrier oscillator amplitude, f_C is the carrier frequency, A_M is the modulator amplitude, and f_M is the modulator frequency

FM signal produced according to (3.2) has a discrete spectrum with one centre frequency F_C and a series of *sidebands* uniformly spaced around it with a frequency interval F_M (Fig. 3.2). Thus all partial frequencies meet the following condition [61]

$$f_P = f_C + k f_M, \quad k \in \mathbb{Z} \tag{3.3}$$

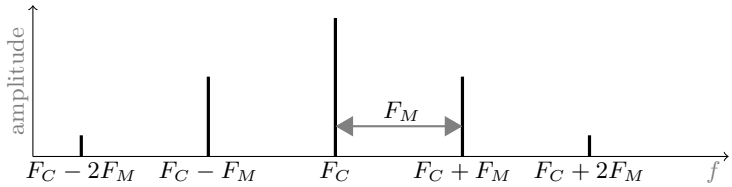


Figure 3.2. Characteristic structure of partial frequencies in a simple FM instrument

Although FM synthesis may be implemented in analogue form, it was originally developed as a digital technique with two crucial advantages over other digital methods popular at that time: the computational efficiency[1], and the ability to produce rich, time-varying spectra. Combination of these qualities allowed to design affordable synthesizers capable of real-time control over complex sounds.

[1] A simple one-carrier, one-modulator instrument requires only two multiplications, one addition, and two table lookups – to retrieve sine values for oscillators.

A simple FM instrument, with one carrier and one modulator, can produce rich spectra, but is obviously not powerful and flexible enough for real musical applications. However, due to general simplicity of the FM principle, designing more complex instruments is relatively straightforward. The most important extensions include:

- adding EGs to control selected synthesis parameters in order to produce time-varying sounds,
- introducing additional oscillators, both as carriers and modulators,
- creating feedback circuits,
- grouping simple synthesis elements into abstract functional units to allow experimenting with their reconfigurations, in a manner similar to modularisation of subtractive synthesizers.

Peculiarly, the high efficiency of FM synthesis is also, indirectly, a source of its main flaw. Since the whole synthesis process is controlled through a few parameters only, each of these parameters has a considerable impact on a timbre of produced sound. Their relation to signal spectrum is relatively complex, and signal details cannot be freely adjusted, which makes controlling signal properties unintuitive. As a result, using FM synthesizer to obtain desired sounds often involves a trial and error approach, and stipulated effect may not be attainable at all.

3.1.1.1. Frequency and Pitch

Frequency Ratio

Formula (3.3) expresses frequencies of partials produced by a simple FM instrument. Partial positions depend on both, F_C and F_M, with F_C constituting the centre of the spectral structure and F_M being the inter-partial frequency interval. Obviously, in a musical instrument frequencies will have to be changed while producing different pitches. In order not to change the spectral structure each time a new pitch is played, both frequencies have to be changed accordingly. Hence a parameter controlling their ratio is introduced [470]

$$R_f = \frac{f_C}{f_M} \tag{3.4}$$

If only a pitch is to be changed, and not a timbre, the ratio has to remain fixed. Often a symbol $c : m$ is utilised instead of R_f.

All FM spectra are discrete, but not all are harmonic (Fig. 3.3). The latter are produced when R_f is a rational number [116]

$$R_f = \frac{N_1}{N_2}, \quad N_1, N_2 \in \mathbb{Z} \tag{3.5}$$

Furthermore, in order for the produced harmonic spectrum to evoke a sensation of one, definite pitch, both N_1 and N_2, after dividing out their common factors, shall

be relatively small. Otherwise resultant f_0, while present, may be located way below audible range, and produced spectrum may be virtually inharmonic. More details regarding impact of R_f on a signal spectrum can be found in the work of Truax [561].

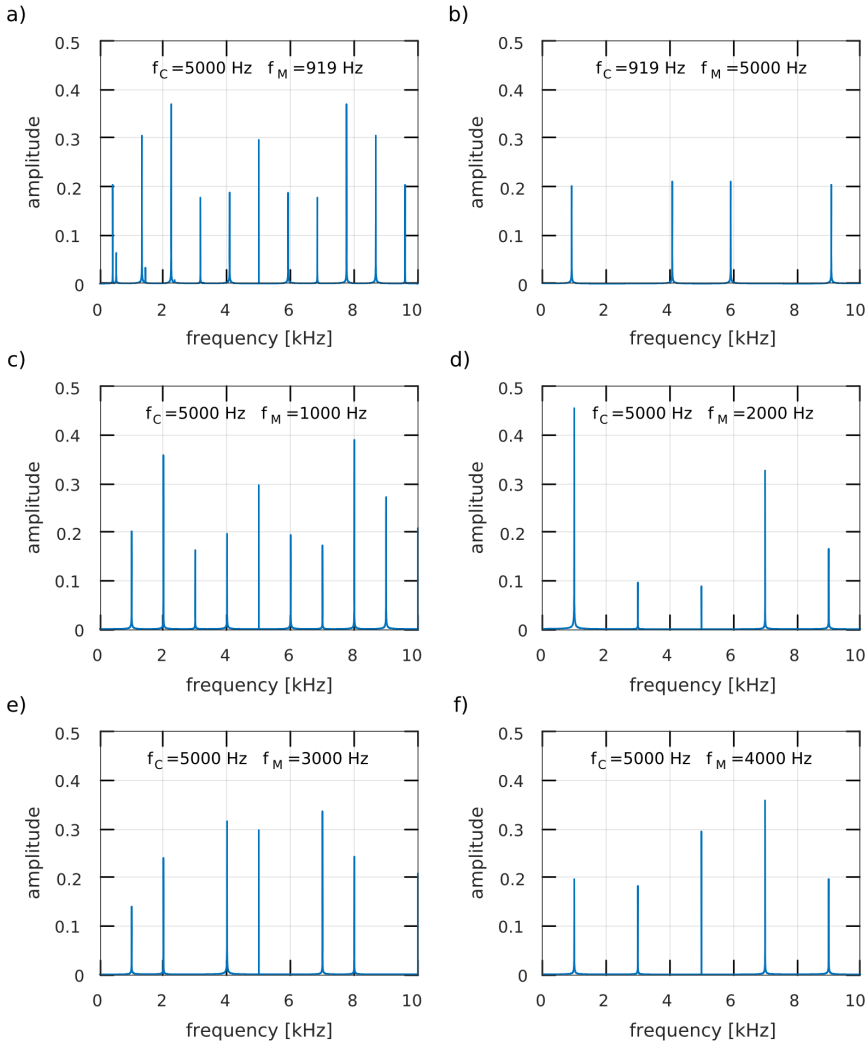


Figure 3.3. The effect of carrier to modulator frequency ratio R_f on FM signal spectrum; lack of symmetry around f_C and presence of small partials around the main ones is caused by partials with negative frequency being reflected from $f = 0$ axis; in (a) and (b) spectra are virtually inharmonic (actually harmonic, though with $f_0 = 1$ Hz); in (c) N_2 (3.5) can be reduced to 1, hence all the harmonics are present; in (d) and (f) reduced N_2 is even, therefore the spectra are odd; in (e) N_2 can be reduced to 3, which results in every third harmonic removed

Fundamental Frequency

With two controllable frequencies it is not immediately obvious how to determine and control pitch of the FM signal. If in (3.5) N_1 and N_2 have their common factors divided out, the fundamental frequency of harmonic FM signal is calculated as

$$f_0 = \frac{f_C}{N_1} = \frac{f_M}{N_2} \tag{3.6}$$

As Chowning points out regarding harmonic FM spectra [116], value N_1 is the harmonic index of the carrier frequency, and value N_2 controls presence of particular groups of harmonics. Specifically, for $N_2 = 1$ all harmonics are present, and $f_0 = f_M$. For even values of N_2 the spectrum is odd, and if $N_2 = 3$, every third harmonic is removed (Fig. 3.3). It has to be considered though, that amplitudes of partials are further affected by the peak frequency deviation A_M, therefore some of remaining partials may still be attenuated or entirely cancelled.

3.1.1.2. Modulation Index

The amount of modulation may be controlled through the modulator amplitude A_M, also referred to as the peak frequency deviation. However, it is a frequency quantity, and musical applications deal with logarithmic frequency scale. As a result, using the same value of A_M in various frequency regions would have a distinctly different auditory effect. Therefore a parameter that controls modulation should scale with frequency. Such a parameter is the ratio of the peak deviation of modulation to the modulating frequency, referred to as the *modulation index* [470]

$$I = \frac{A_M}{F_M} \tag{3.7}$$

Value of $I = 0$ represents a case without modulation – the output is entirely a carrier signal. Even though the name of the parameter might suggest otherwise, I can assume not only natural values, but real as well. The effect of changing value of I on a signal spectrum is presented in Figure 3.4.

Due to frequency scaling, modulation index is a better match for perceptual changes in timbre than a direct modulator amplitude, and as such it is more commonly used as a parameter. Therefore it is convenient to replace A_M with I in (3.2) [61]

$$u(t) = A_C \sin\left(2\pi \int_0^t \left(f_C + I f_M \cos\left(2\pi f_M t'\right)\right) dt'\right) \tag{3.8}$$

Formula (3.8) may be written in a less strict, simplified form [61]

$$u(t) = A_C \sin\left(2\pi f_C t + I \sin\left(2\pi f_M t\right)\right) \tag{3.9}$$

The difference is, that while in (3.8) it is the frequency that is being modulated, in (3.9) it is the phase. The issue is further discussed in works of Bate and Holm [41, 242].

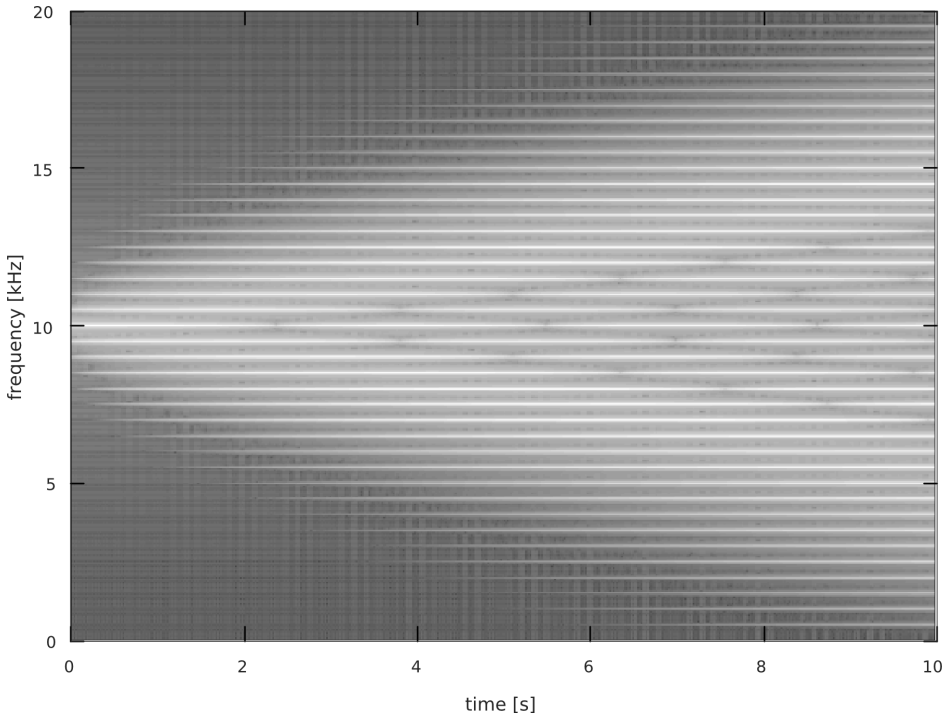


Figure 3.4. The effect of increasing modulation index I on FM signal spectrum; values of I are set to match time in seconds, i.e. $I = 1$ in $t = 1$ s, etc.; larger values of I produce wider spectrum; it is important though to note, that amplitudes of partials do not increase monotonically with I, but can decrease and reach 0 in certain points

Modulation index allows to roughly estimate two spectral characteristics: the number of significant sideband pairs[2], and the bandwidth of the FM signal. The former, according to De Poli [154] is approximately $I + 1$, while the latter, according to Chowning is expressed as [116]

$$BW_{FM} \approx 2f_M(I + 1) \tag{3.10}$$

As Roads points out [470], modulation index that controls signal bandwidth may be convenient in simulation of musical instrument behaviour. Specifically, in acoustic instruments a bandwidth often tends to widen when an amplitude of a signal increases. In FM it can be efficiently simulated by controlling both parameters, i.e. A_C and I, through a common envelope, or a pair of similar envelopes. Extension of a simple FM instrument, with controllable modulation index I, ratio R_f, and two EGs, is presented in Figure 3.5. For the purpose of facilitating finer control over timbre evolution, commercial FM synthesizers apply EGs with additional segments (Fig. 3.6).

[2]In this case partials with amplitudes larger than 1% of the amplitude of the carrier are considered significant.

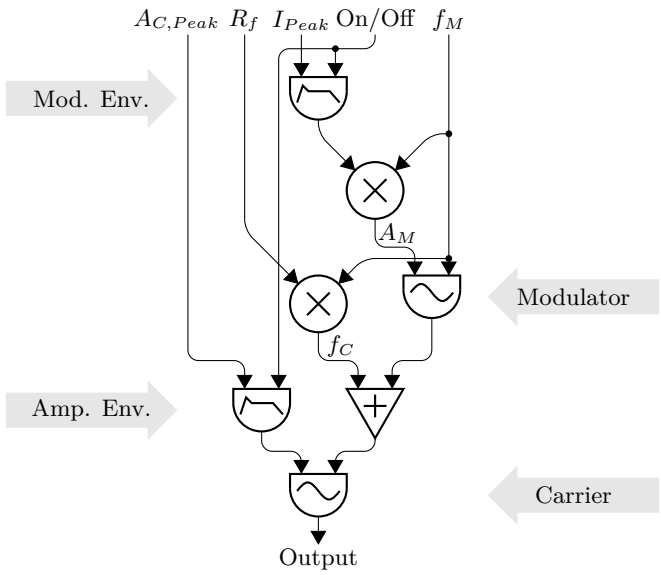
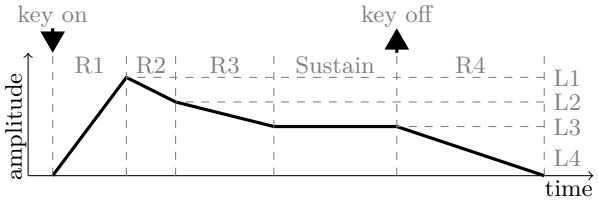


Figure 3.5. A simple FM instrument with two EGs; the first EG controls output signal amplitude with user-set peak value $A_{C,Peak}$; the second EG controls modulation index, with user-set peak value I_{Peak}; EGs are triggered by a common On/Off signal; f_C is controlled indirectly, by setting f_M and ratio R_f, which allows to change pitches while keeping the signal spectrum harmonic

a)



b)

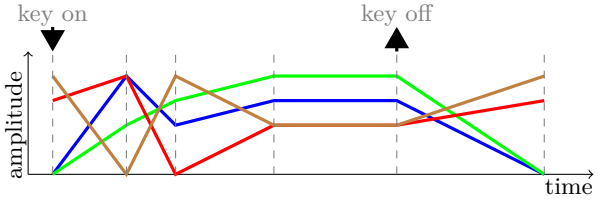


Figure 3.6. Envelope used in commercial Yamaha FM synthesizers (a) and some of its possible shapes (b); the four segments (R1–4) and levels (L1–4) are freely adjustable, so that inverted envelopes may be designed as well
Source: author's elaboration, based on Russ [485]

Even though in general a bandwidth of the FM signal increases with I, with subsequent sideband pairs emerging (Fig. 3.4), amplitudes of partials vary with I in a non-monotonic manner (Fig. 3.7), and for some values of I amplitudes of certain sidebands reach 0. Such behaviour makes predicting a spectral envelope, and hence a timbre, an unintuitive task.

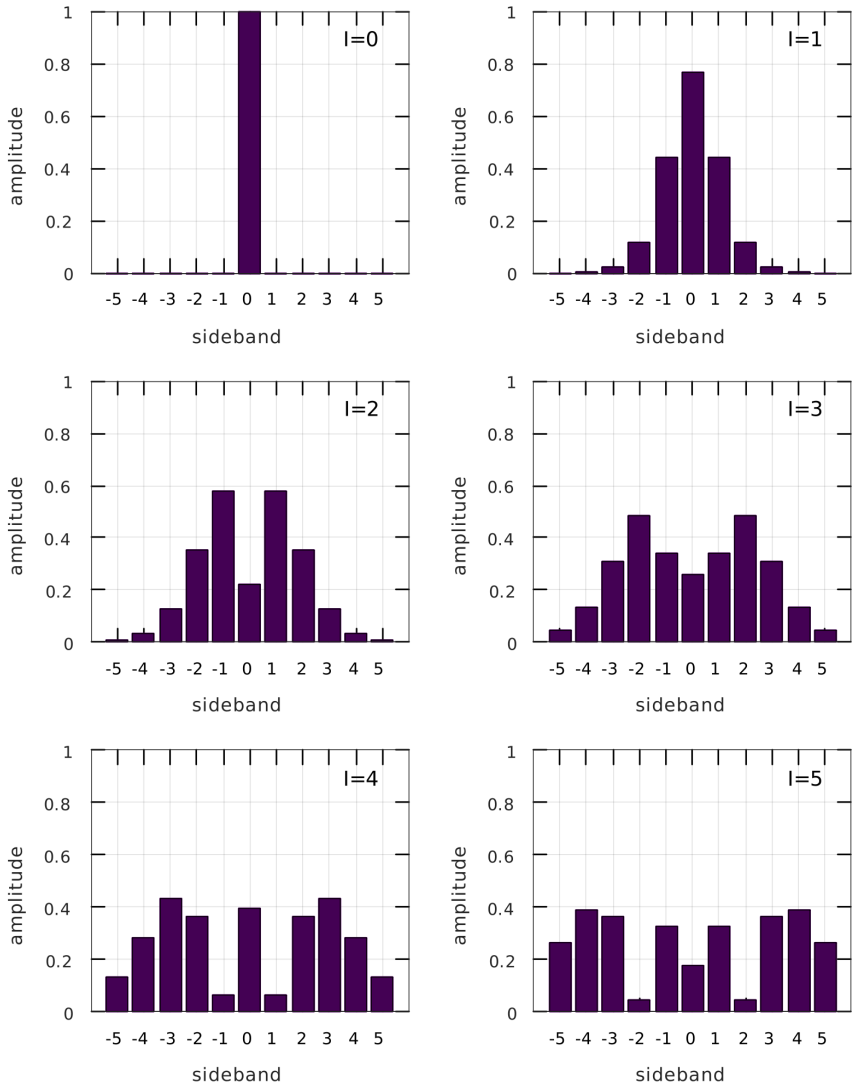


Figure 3.7. Amplitudes of first five pairs of sidebands in FM signal spectrum for increasing value of modulation index I

It is possible to determine partial amplitudes by developing (3.9) into the following form [116, 154, 51]

$$u(t) = \sum_{k=-\infty}^{\infty} J_k(I) \sin|2\pi \left((f_C + k f_M)\, t \right)| \tag{3.11}$$

where J_k are the Bessel functions of the first kind and order k (Fig. 3.8), and k is the index of sideband. Indices start from 0 in the carrier position f_C, and continue in both directions, which can be observed as a spectral symmetry around f_C. Therefore Bessel function $J_k(I)$ is the amplitude of partial $f_C + k f_M$ for a particular value of I (Fig. 3.9).

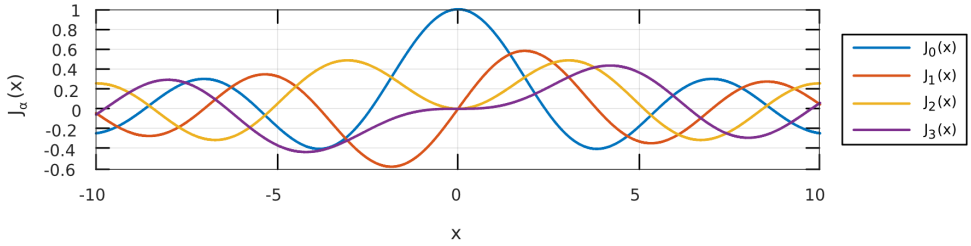


Figure 3.8. Bessel functions of the first kind and orders (α) from 0 to 3

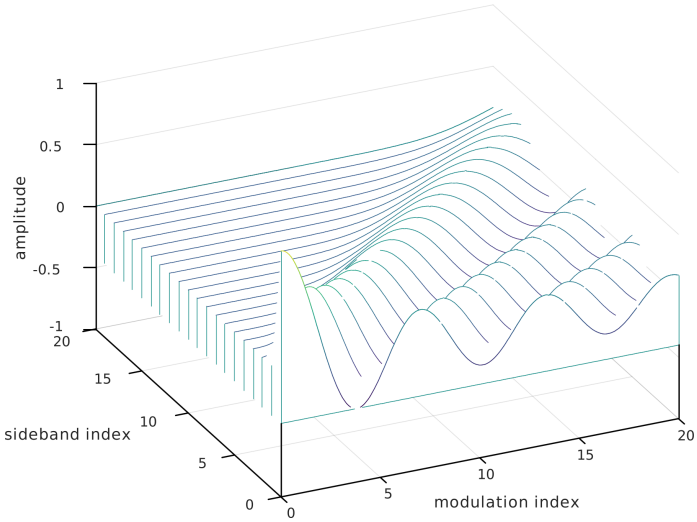


Figure 3.9. Amplitudes of FM partials in function of a sideband index (Bessel function order) and a modulation index I (Bessel function argument); lines represent evolution of partial amplitudes while increasing modulation index; a cross-section along a particular value of I represents a discrete spectrum with amplitudes of subsequent sidebands

As can be seen in Figure 3.8, J_α assume not only positive, but negative values as well. Moreover, Bessel functions of the first kind have the following property

$$J_{-n}(x) = (-1)^n J_n(x), \quad \text{for } n \in \mathbb{Z} \tag{3.12}$$

As an effect, even order Bessel functions have the axial symmetry around $x = 0$, while odd order functions have the central symmetry around $(x, y) = (0, 0)$.

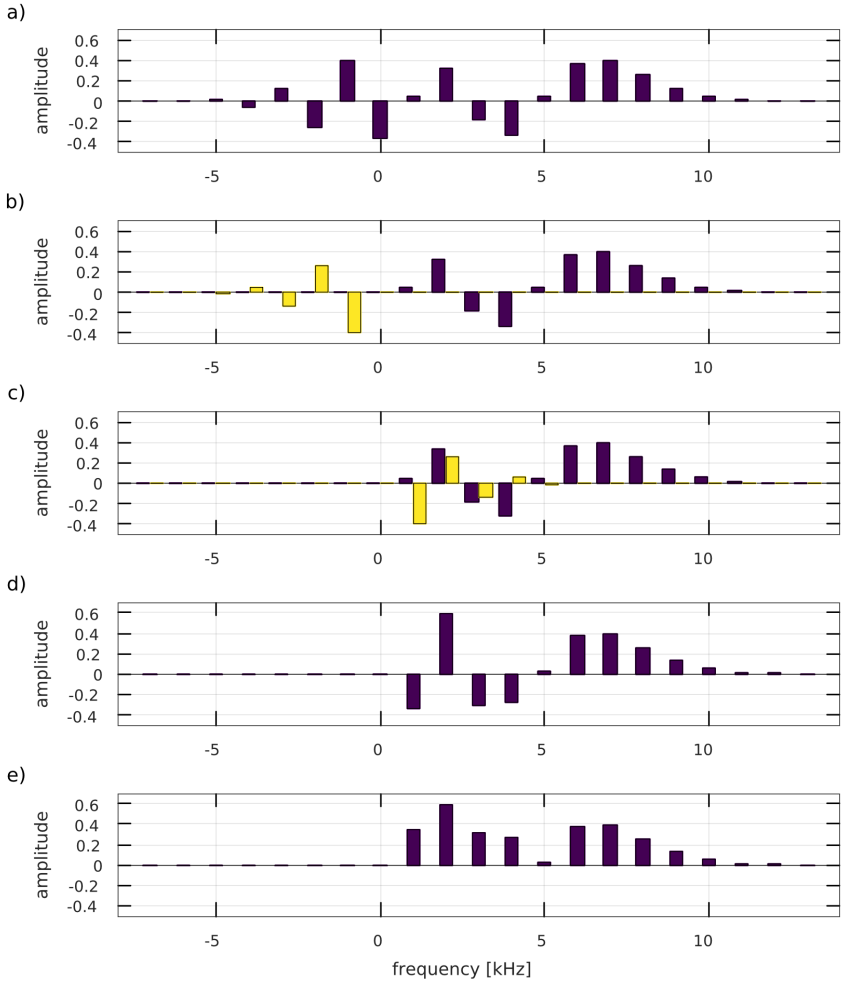


Figure 3.10. An impact of inverted phase on a spectrum of FM signal with $f_C = 3$ kHz, $f_M = 1$ kHz, and $I = 5$; in (a) some partials have their phases inverted, according to (3.11), which is represented by negative amplitudes; in (b) partials below $f = 0$ have their phases inverted, and in (c) they are reflected above $f = 0$; in (d) partials with corresponding frequencies are added, considering phases, and in (e) phase information is disregarded to display only resultant partial amplitudes

While negative values of J_α do not affect absolute amplitude values, and as such are sometimes disregarded, as in Figure 3.7, they affect phases of partials, which is of consequence when two partials occupy the same frequency, for instance, if one of them has been reflected from $f = 0$. Due to this reason in schematic plots of FM spectra it is common to represent partials with inverted phase, i.e. negative amplitude, by downward bars, below the frequency axis [116, 470], as in Figure 3.10.

Unlike rapidly varying amplitudes of individual sidebands, the overall FM signal level varies only to a small degree when modulation index is changed. It is a convenient property with regards to control over musical dynamics, making an overall signal level virtually independent from the timbre control. Dynamics of the FM signal may therefore be controlled by simple adjustments of A_C, possibly through an envelope.

Reflected Partials

The FM signal produces an infinite number of sidebands around a carrier frequency. Even though their amplitudes drop below a significant level at some distance from a carrier frequency, due to bandlimited nature of a digital signal a certain number of significant sidebands often exceeds lower or upper frequency limit. When this happens, these sidebands are reflected and their phases are inverted. Usually it is the lower limit that is exceeded first, as presented in Figure 3.10.

Even though reflected partials make designing a specified timbre more demanding and less intuitive, the outcome is generally advantageous. Unless partials cancellation occurs in the majority of cases, partials reflected from $f = 0$ boost lower part of spectrum, which otherwise would be weaker in comparison to surroundings of f_C. Without reflected partials FM spectrum is symmetric around f_C. Breaking this symmetry allows to produce more instrument-like sounds.

3.1.1.3. Multiple Carriers and Modulators

A simple FM, with one carrier and one oscillator, is very efficient, yet it allows quite limited control over produced spectra. In order to provide further control parameters, and allow to shape the signal in a more detailed way, additional oscillators are introduced, either as carriers, or modulators.

In **multiple-carrier FM** (MCFM) one common modulator is connected to inputs of a few carriers (Fig. 3.11). Such arrangement of oscillators produces a spectrum that is a superposition of spectra of all modulated carriers [470]

$$u(t) = \sum_{i=1}^{N} A^{w_i} \sin\left(2\pi f_{C_i} t + I_i \sin\left(2\pi f_M t\right)\right) \tag{3.13}$$

where N is the number of carriers, $A \in (0, 1]$ is the amplitude constant, w_i are the weighting coefficients, f_{C_i} are the carrier frequencies, I_i are the modulation indices, and f_M is the single modulator frequency. It is common to set $f_M = f_{C_1}$ [116, 470].

Since a single carrier-modulator set produces a centred spectrum that can form a peak, employing a number of such sets MCFM is able to produce spectrum with formant regions, and control each of them separately (Fig. 3.12). Carriers can also have

separate amplitude envelopes to simulate various decay times in different frequency regions, which is a typical feature of instrument spectra.

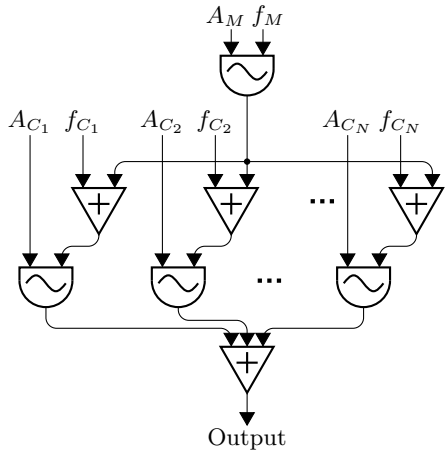


Figure 3.11. Multiple-carrier FM

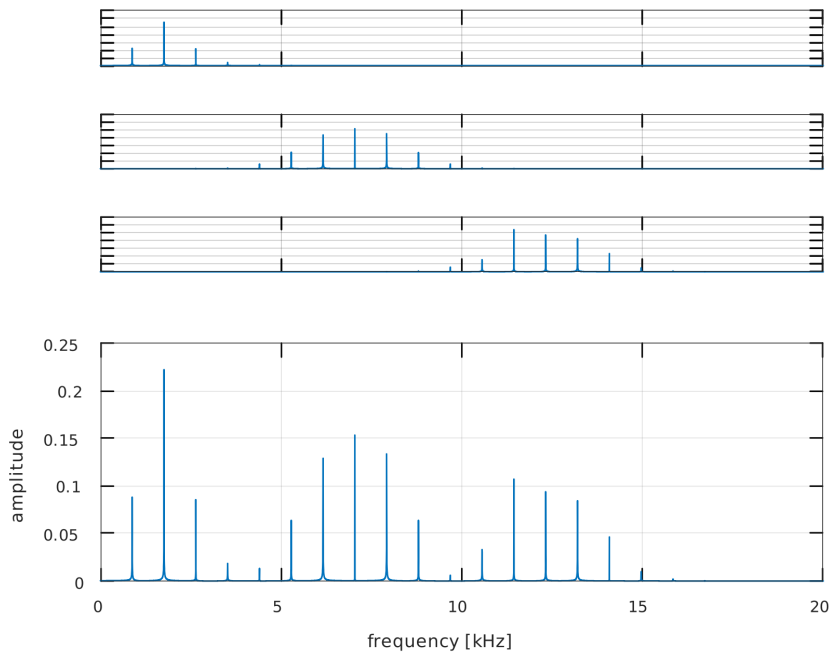


Figure 3.12. Spectrum of a three-carrier MCFM signal (bottom plot) with the following parameters: $f_M = f_{C_1} = 880$ Hz, $f_{C_2} = 7040$ Hz, $f_{C_3} = 12320$ Hz, $I = 1.5$; three top plots present separate spectra of each of carrier-modulator subsets without weights

If a few modulators is connected to one carrier, such arrangement is referred to as **multiple-modulator FM** (MMFM). Modulators can be connected in parallel or in series. Both configurations are presented in Figure 3.13.

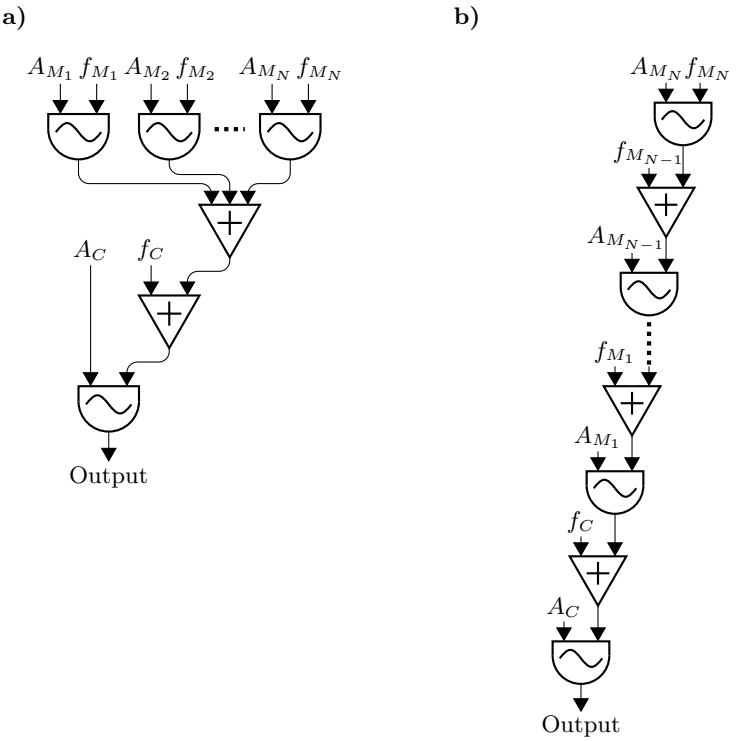


Figure 3.13. Multiple-modulator FM in parallel (a) and serial (b) configuration

In case of **parallel MMFM** the expression for partial frequencies changes from a simple case of (3.3) to the following form [470]

$$f_P = f_C + \sum_{i=1}^{N} k_i f_{M_i}, \quad k_i \in \mathbb{Z} \tag{3.14}$$

where N is the number of modulators. Given two modulators, it can be interpreted as each of the sidebands produced by the modulator closer to the carrier is becoming an additional carrier, with its own sidebands produced by the second modulator. Roads refers to this phenomenon as the explosion in the number of partials [470], since adding subsequent modulators increases this number exponentially (Fig. 3.14). For comparison, in MCFM number of partials increases linearly with the number of carriers.

Output signal of the parallel MMFM is given by the following formula [327, 495]

$$u(t) = A_C \sin\left(2\pi f_C t + \sum_{i=1}^{N} I_i \sin\left(2\pi f_{M_i} t\right)\right) \tag{3.15}$$

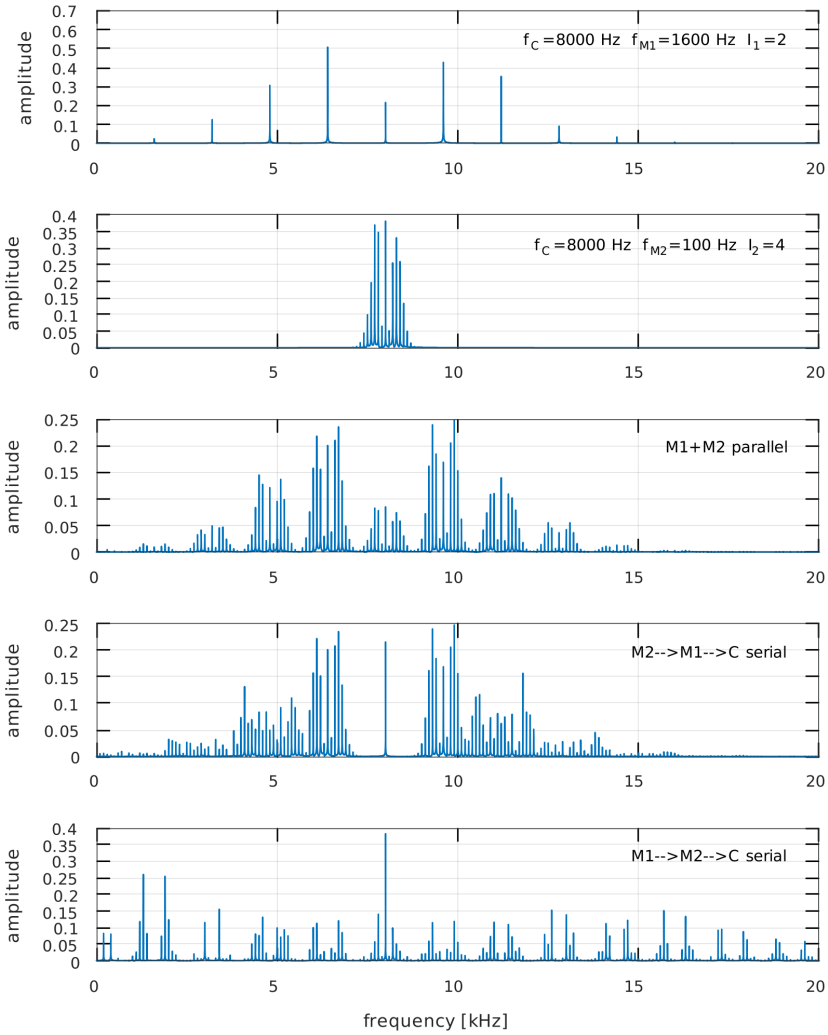


Figure 3.14. Spectra of two-modulator MMFM signals (three bottom plots); first two plots present a simple FM with one carrier and one modulator, M1 and M2, respectively, with two different modulation indices I1 and I2; the third plot is a spectrum of parallel MMFM with modulators M1 and M2; two last plots present spectra of serial MMFM with two possible oscillator orders; in the fourth, M2 modulates M1, and M1 modulates the carrier, while in the fifth, M1 modulates M2, and M2 modulates the carrier

Serial MMFM may produce an immense number of components [470]. Even in the simplest case of two modulators signal spectrum is complicated – each of carrier's sidebands is both, modulated, and a modulator [553] (Fig. 3.14). The signal of serial MMFM can be produced according to the following nested formula [495, 470]

$$u(t) = A_C \sin\left(2\pi f_C t + I_1 \sin\left(2\pi f_{M_1} t + I_2 \sin\left(2\pi f_{M_2} t + \ldots\right)\right)\right) \tag{3.16}$$

3.1.1.4. Feedback

A simple FM with a pair of oscillators may produce signals with unique spectra of a 'synthetic' quality, but it struggles to reproduce sounds of most of acoustic instruments. The problem has two sources. Firstly, the spectrum is symmetric around f_C, but it can be simply remedied by producing partials with 'negative' frequencies and reflecting them back. Secondly, bandwidth of the signal is controlled through the modulation index, and changes to this parameter cause large fluctuations in partial amplitudes – a phenomenon that does not occur in acoustic instruments. The second issue can be solved by introducing feedback.

In the **feedback FM** (FFM) output signal is fed back into the frequency input. The simplest configuration utilises only one oscillator (Fig. 3.15) and produces a signal that can be calculated according to the following formula [470]

$$u(t) = \sum_{k=1}^{\infty} \frac{2}{k\beta} J_k(k\beta) \sin\left(k 2\pi f t\right) \tag{3.17}$$

where k is the partial index, J_k are the Bessel functions of order k, and β is the feedback factor. In one-oscillator FFM the product $k\beta$ functions as a modulation index. It increases with partial index. Additional scaling coefficient $\frac{2}{k\beta}$ is introduced to decrease amplitude of partials when their indices increase, which can be observed in spectra presented in Figure 3.16. Due to partial amplitudes decreasing towards higher frequencies and more predictable behaviour of the spectral envelope with increasing signal bandwidth, FFM is better suited to simulate broader class of acoustic instrument sounds than a simple FM instrument from Figure 3.1.

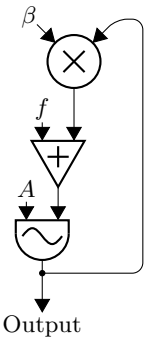


Figure 3.15. A feedback FM instrument; parameter β is the feedback factor

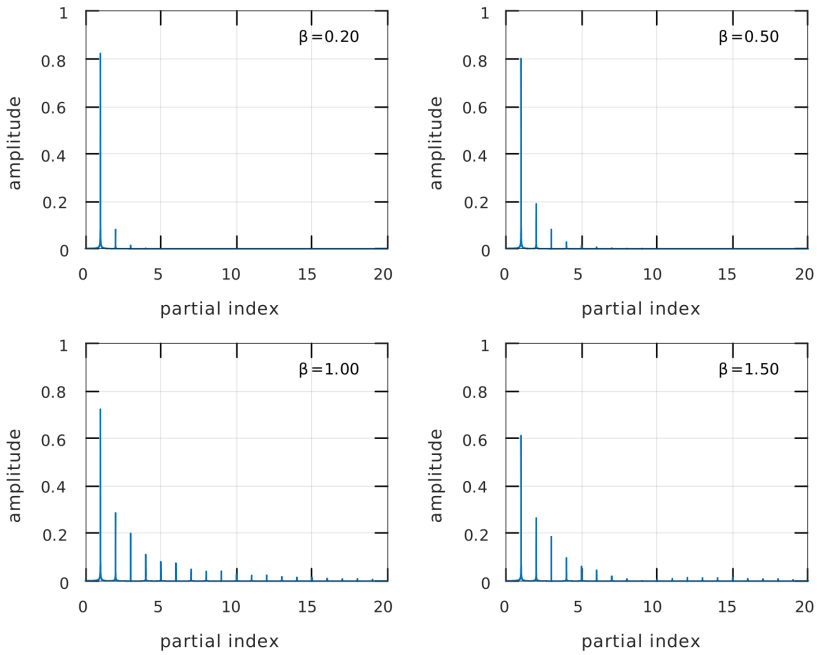


Figure 3.16. Spectra of the one-oscillator feedback FM for various values of feedback factor β

Output of a one-oscillator FFM can be directed to modulate another oscillator, with parameter M being the modulation index. Such configuration is referred to as two-oscillator FFM, and is presented in Figure 3.17a. Example settings of its parameters are given in Table 3.1. Figure 3.17b presents configuration of a three-oscillator indirect FFM. It can be utilised to produce non-pitched sounds by setting f_1, f_2, and f_3 to non-integer ratios.

Table 3.1. Selected parameter values for two-oscillator FFM (Fig. 3.17a) and their effect, according to Roads [470]

Parameters	**Effect**
$M \in [0, 2]$	Monotonically decreasing spectrum
$\beta > 1$	Increase in overall amplitude of higher-order partials – the effect of a variable filter
$M = 1$, $f_C = f_M$	Spectrum of the one-oscillator FFM
$f_C/f_M = 2$, $M = 1$, β varies in $[0.09, 1.571]$	Variation between quasi-sine and quasi-square

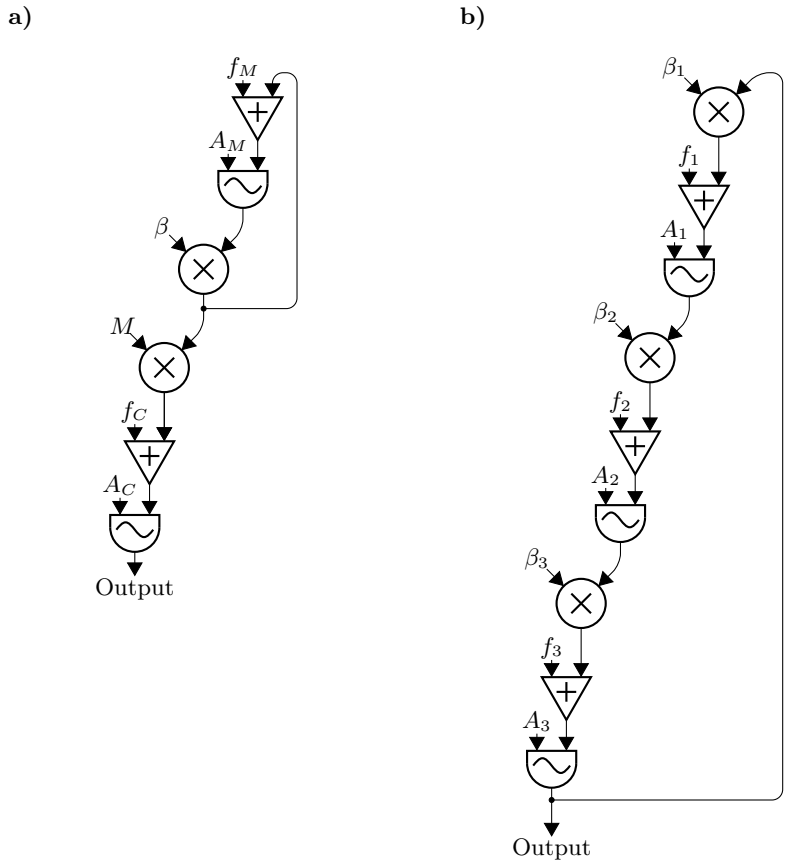


Figure 3.17. Two-oscillator feedback FM (a) and three-oscillator indirect feedback FM (b)

3.1.1.5. Operators and Algorithms

FM configurations such as MCFM, both variants of MMFM, various FFM arrangements, as well as other multi-oscillator or feedback configurations, commonly referred to as complex FM, may be presented in a modularised, symbolic manner with the use of abstractions referred to as *operators* and *algorithms*. The operator is a single FM module. It includes an oscillator, an amplifier, and an envelope generator, as shown in Figure 3.18. Output of one operator can be connected to a modulation input of another one. An arrangement of connected operators is referred to as the algorithm.

Most FM synthesizers allowed to use between four and eight operators. A very popular Yamaha[3] DX7 model utilised six-operator algorithms [51], a GS1 model used eight [485], and an OPL3 chip integrated in various computer sound boards provided both four, and two-operator algorithms.

[3]Yamaha patented FM method in 1981 [559].

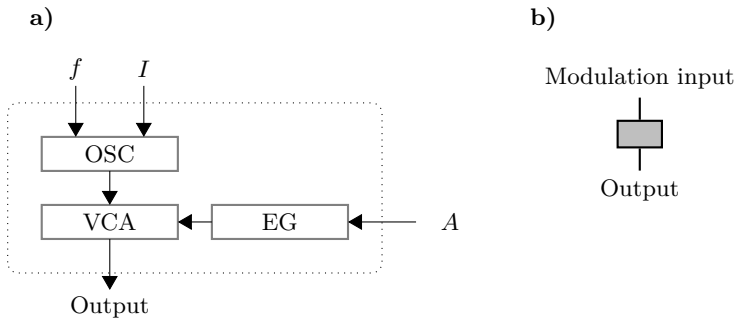


Figure 3.18. Elements of an FM operator (a) and its symbol used in synthesizers (b)

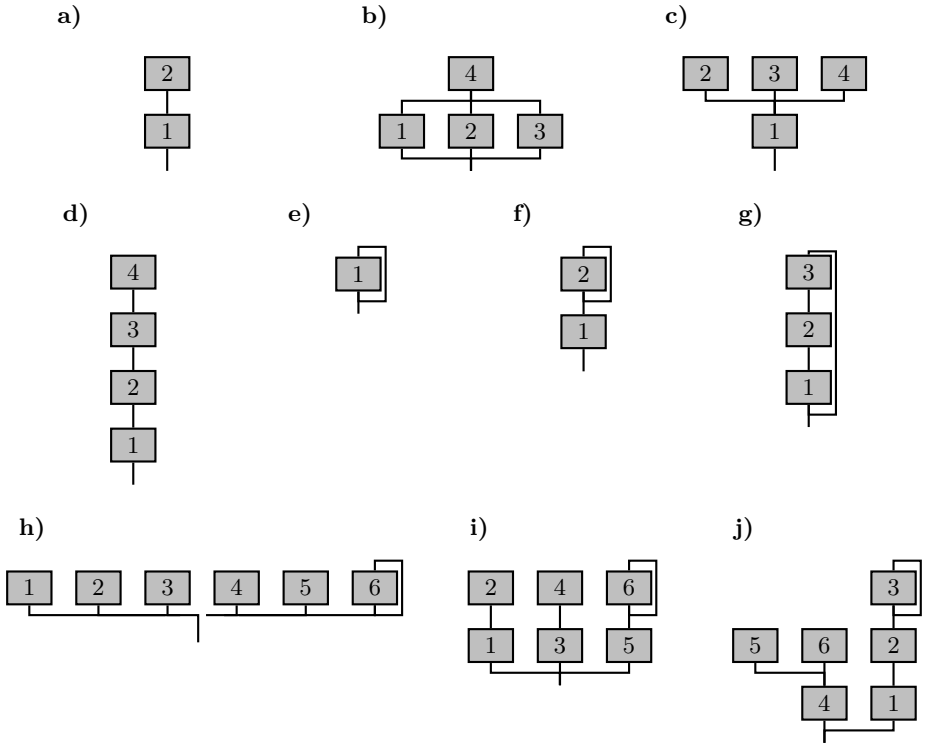


Figure 3.19. FM algorithms: a) a simple FM; b) MCFM; c) parallel MMFM; d) serial MMFM; e) one-oscillator FFM; f) two-oscillator FFM; g) three-oscillator indirect FFM; h) additive with feedback; i) pairs with feedback; j) combination; last three algorithms (h–j) were utilised in the Yamaha DX7 FM synthesizer
Source: author's elaboration, based on Benson [51]

Even though with operator-algorithm approach FM synthesis became modular and potentially very flexible in creating user-defined configurations, the actual synthesizers were more limited. The most popular models only allowed to choose from some number of factory-set algorithms, e.g. Yamaha DX7 had 32 of them. Figure 3.19 presents algorithms for the basic forms of FM, as well as three of six-operator algorithms of Yamaha DX7. Facilities to create user defined algorithms were limited to more advanced models.

In actual synthesizers operators have various sets of capabilities. The most common differences are the number of controlled parameters, such as envelope segments, and precision of control over their values. In more advanced applications not only sine oscillators, but other waveshapes are available as well.

3.1.1.6. Simulation of Instruments and Resynthesis

FM synthesis is mostly regarded for its distinctive 'synthetic' sound, and less for realistic reproduction of acoustic instruments. However, there are instructions regarding particular parameter values that result in an instrument-like sounds. Chowning [116] lists a few of such sets for a simple FM instrument, as shown in Table 3.2. In pitched instruments R_f parameter can be slightly detuned from the exact value, e.g. by 0.5 Hz. It will result in a beating effect. Furthermore, a detuning of R_f can be controlled by the amplitude or modulation envelope, allowing it to evolve as an additional timbre-related quality.

Table 3.2. Parameter values for a simple FM instrument that produce instrument-like sounds, based on Chowning [116] and Roads [470]; modulation index varies in a given range controlled by the envelope generator

R_f	I	**Instrument**
1	$[0, 7]$	Brass-like
3	$[0, 2]$	Woodwinds and organ-like
5	$[0, 1.5]$	Bassoon
1.5	$[2, 4]$ (EG inversed)	Clarinet
1.4	$[0, 10]$	Bell-like
1.4	$[0, 2]$	Drum-like
$\frac{16}{11}$	$[0, 25]$	Wood drum
$2^{\frac{1}{2}}$ and other irrational	various	Various percussive and bell-like

MCFM with two carriers was initially applied to simulate sound of a trumpet and other brass instruments by Morrill [389]. The first carrier produces the fundamental and its surroundings, while the second carrier is fixed at the main formant region of the instrument. With the addition of periodic and random *vibrato* effect Chowning was able to apply MCFM to the synthesis of a soprano and bass voice [118, 119].

A simulation of piano sound was produced by Schottstaedt by applying MMFM with two modulators [495]. Value of f_{M_1} was set very close to the value of f_C, in order to introduce a certain amount of spectral inharmonicity inherent for piano sound

[67, 27, 47]. As for the second modulator, $f_{M_2} \approx f_{M_1}$. Value of modulation index decreases as f_C increases, to simulate simpler spectrum of higher pitched sounds.

Sounds that may be categorised as string-like were produced using MMFM with three modulators, with frequency-dependent values of particular modulation indexes [495]. A combined MCFM and MMFM method allowed to simulate a deep bass voice [118, 119].

There is no straightforward way to carry out a resynthesis process using FM method, and initial attempts at synthesising instrument-like sounds involved manual trial and error. One of the main problems is the requirement to encode an arbitrary spectrum, possibly evolving, into a very limited set of parameters. Early works aimed at estimating parameters for steady state signals [275], for instance, by applying nonlinear analysis based on an extended Kalman filter [463]. Further research led to solutions involving evolving sounds as well [310, 249], including optimisations based on genetic algorithms [250, 247, 546, 335, 248]. A block diagram for the evolutionary FM matching procedure applied by Mitchell [377] is presented in Figure 3.20.

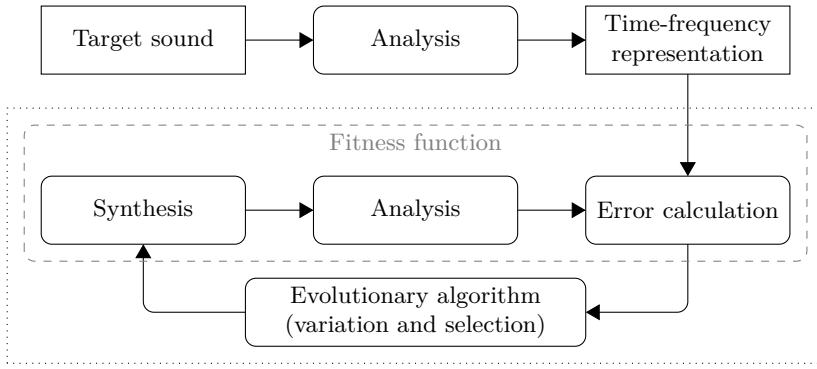


Figure 3.20. Evolutionary sound matching model
Source: author's elaboration, based on Mitchell [377]

3.1.1.7. Variants and Derivatives of FM Synthesis

FM With Non-Sinusoidal Signals

In basic FM synthesis implementations it is commonly assumed that carriers and modulators are sine signals. Even though it simplifies the process of sound production and makes it more predictable, it is not a requirement. Any signal, either abstract or sampled, can modulate or be modulated. However, additional spectral components in carrier or modulator lead to considerable increase in complexity of the resulting FM spectrum. All partials of a carrier or a modulator take part in modulation, and may be considered separate sine carriers or modulators. In effect, a large number of sidebands is produced. Many of them share a common frequency, particularly in case of harmonic frequency relations. Since Bessel functions of the first kind J_α, controlling partial amplitudes, can assume negative values, and some partials have their phase inverted due to being reflected from bandlimits, a complex pattern of

partial additions and cancellations may emerge. Due to its discrete character, the same principle applies to a digital bandlimited noise. A relatively simple example is presented in Figure 3.21.

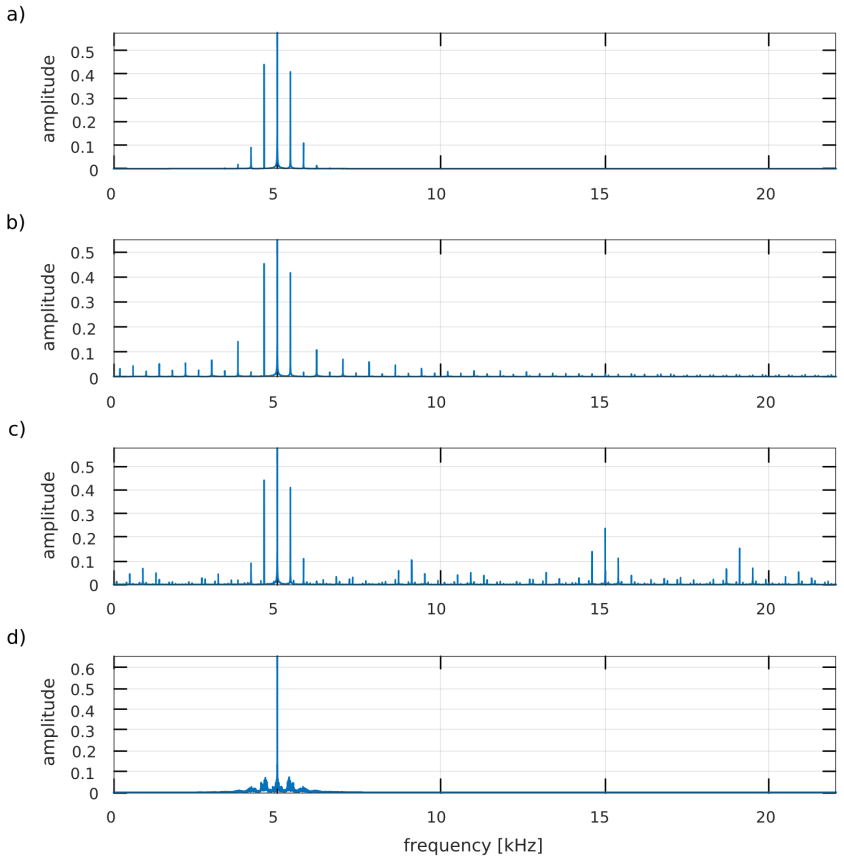


Figure 3.21. Spectra produced using non-sinusoidal carrier and modulator; in all cases the sampling frequency $f_s = 44.1$ kHz; plot (a) is the basis for comparison, presenting FM with sine carrier ($f_C = 5000$ Hz) and sine modulator ($f_M = 400$ Hz), $I = 1$; in (b) sine modulator has been exchanged for a square signal ($f_0 = 400$ Hz); in (c) modulator is sinusoidal ($f_M = 400$ Hz), but the carrier is square ($f_0 = 5000$ Hz); due to sidebands extending far in both directions, noticeable partial reflections occur on bandlimits, and some of them produce inharmonic components as a result of chosen f_s value; in (d) sine signal is modulated with bandlimited noise (300–500 Hz), and $I = 4$

Timoney et al. give the following formula for the FM signal with a sine carrier modulated by a signal that has a discrete spectrum [553]

$$u(t) = \sin\left(2\pi f_C t + \sum_{k=1}^{K} I_k \sin\left(2\pi k f_M t + \phi_k\right) + \theta\right) \tag{3.18}$$

where the modulator is expressed by its Fourier series, K is the number of modulator partials, I_k is the modulation index of partial k, f_M is the fundamental frequency of the modulator, ϕ_k is the phase shift of the partial k, and θ is the carrier phase shift. Instead of various I_k values, all modulator partials can share a common value I.

When a sine signal modulates a carrier that has a discrete spectrum, the expression for a resulting signal assumes the following form [553]

$$u(t) = \sum_{k=1}^{K} \sin\left(2\pi k f_C t + I \sin\left(2\pi f_M t\right) + \phi_k\right) \tag{3.19}$$

Formula (3.18) can be rewritten using Bessel functions to indicate individual partials in the resulting signal [495, 327]

$$u(t) = \sum_{k_K} \cdots \sum_{k_1} \left(\prod_{l=1}^{K} J_{k_l}(I_l) \right) sin \left(2\pi f_C t + \left(\sum_{l=1}^{K} k_l \left(2\pi l f_M t + \phi_l\right) \right) + \theta \right) \tag{3.20}$$

Timoney et al. proposed a software implementation of (3.20) that does not require nested loops, can be generalised, and may be applied in a distributed processing environment.

Exponential FM

In majority of cases FM synthesis is implemented in a digital form. An analogue implementation is also possible, however it leads to an issue regarding frequency scale. In analogue synthesizers VCO commonly responds to a one-volt-per-octave protocol, which assumes linear dependence between the voltage and musical interval in 12-TET system[4]. Such approach makes the relation between the voltage, which is a control signal, and the frequency exponential [470]. Therefore, if one analogue VCO modulates another, output signal of a modulator causes a carrier VCO to oscillate by an equal interval above and below the carrier frequency, which makes the modulation exponential with regards to the frequency. Hence the analogue variant of FM synthesis is referred to as exponential FM.

The consequence of exponential relation between the modulator output and carrier input is shift of the centre frequency. It leads to the auditory effect of pitch detuning. The pitch rises with increasing modulation index. The relation was studied by Hutchins, who presented the formula for exponential FM signal [258]

$$u(t) = \sin\left(2\pi I_0 \left(V_M \ln(2)\right) f_{Ce} t + \left(\sum_{k=1}^{\infty} \frac{2 f_C}{k f_M} I_k \left(V_M \ln(2)\right) \sin\left(2\pi k f_M t\right) \right) \right) \tag{3.21}$$

which is a multi-component complex FM signal, where V_M is the time-varying component of the modulation signal voltage $V(t) = V_0 + V_M \cos(2\pi f_M t)$, and $f_{Ce} = f_C \exp(V_0 \ln(2))$.

[4] 12-tone equal temperament is a musical instrument tuning system that defines a frequency ratio of 2 as an octave interval, and divides this interval into 12 equal semitone intervals, each having frequency ratio of $\sqrt[12]{2}$.

Therefore the actual carrier frequency changes to

$$f_E = f_{Ce} I_0 \left(V_M \ln(2) \right) \tag{3.22}$$

while the frequency deviation of each term k is expressed by

$$D_k = \frac{2 f_C}{k f_M} I_k \left(V_M \ln(2) \right) \tag{3.23}$$

Hutchins suggested a number of solutions to the pitch-detuning problem that may be introduced in analogue implementation, such as adding an external logarithmic amplifier to the modulator output. Timoney and Lazzarini further studied the problem for virtual analogue applications [551].

Phase Distortion

While Yamaha commercialised FM synthesis and patented its particular implementation, the phase modulation [117], another manufacturer attempted to develop an alternative. Casio experimented with implementing the phase distortion (PD) technique [262], which may be regarded as a subset of phase modulation or FM method [554].

In basic PD a table-lookup sine oscillator is scanned with a ratio that varies over a cycle. A phase increment that indexes a sine lookup table is therefore modulated, but in PD carrier and modulator are synchronised per cycle. In Casio implementation *phase transforms*, also referred to as *phase distorters*, are piecewise linear functions. Such function for modulating cosine into a sawtooth waveform takes the following form [554]

$$\phi_{mod}(t) = (1 - 2P)\, \pi \frac{(\mathrm{saw}(t, P) + 1)}{2} \tag{3.24}$$

where $P \in (0, 1)$ is the part of cycle in which the sawtooth is increasing. The sawtooth is given by $\mathrm{saw}(t, P)$. The function is scaled and shifted to fit between 0 and $(1-2P)\pi$. The result for three different values of P is presented in Figure 3.22.

In comparison to simple FM, PD produces spectra that are more linear, hence it can be more easily adapted to simulate spectra of analogue synthesizers or acoustic instruments. However, it is also regarded as less flexible than FM [470].

Further Derivatives

The computational efficiency of FM synthesis makes it a convenient element or basis for other synthesis methods. Slater used cross-coupled FM oscillators in an approach referred to as **chaotic FM synthesis** [514]. The method produces sounds that can vary from basic sine waveforms, through complex discrete spectra, to coloured noise. Interestingly, contrary to most FM synthesis methods, chaotic FM may be conveniently applied in analogue form.

Adaptive FM synthesis (AdFM) [326] proposed by Lazzarini et al., is an attempt to address an issue of control inherent to FM synthesis. When FM synthesizer

is controlled by a physical, instrument-like controller, it is difficult to determine appropriate mapping of gestural control onto synthesis parameters. In AdFM a carrier is modulated by an arbitrary, real-world input signal. Phase modulation is achieved either by using delay lines or heterodyning. Since there is no implicit control over a frequency of the arbitrary signal, it needs to have its parameters estimated before being used as a modulator, to keep required value of R_f. By combining FM with signals of acoustic instruments AdFM can produce hybrid natural-synthetic sounds.

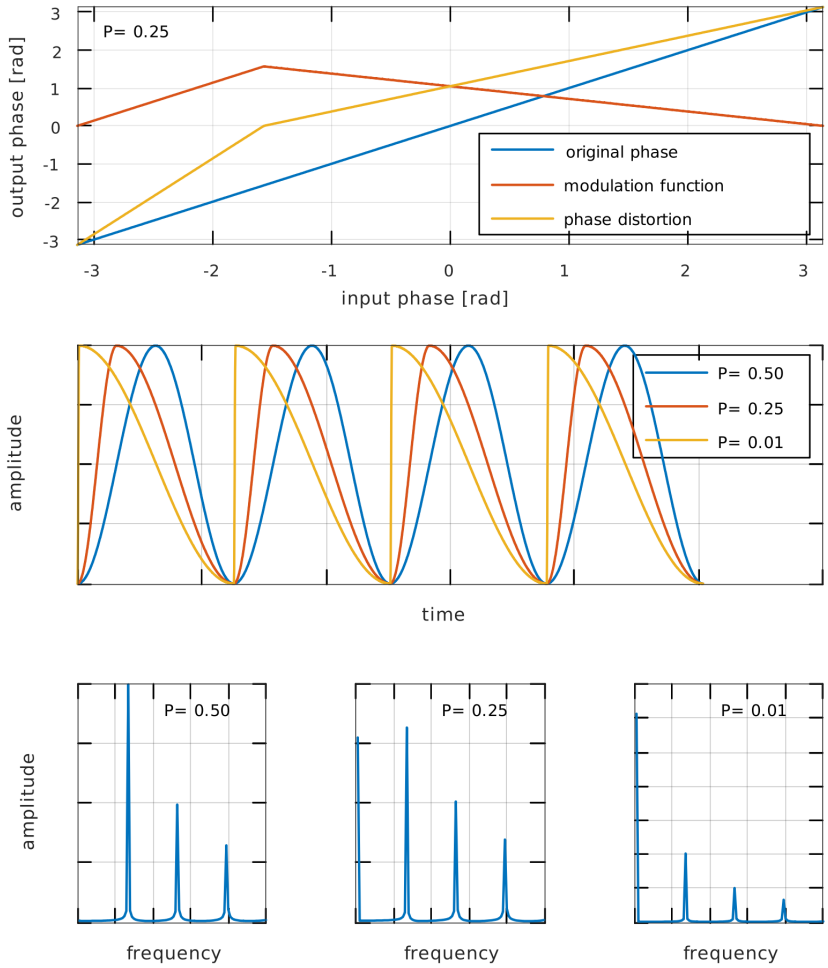


Figure 3.22. Top plot presents the phase distortion function that produces sawtooth-like waveform, according to (3.24); middle plot is the PD output for three values of P; bottom plots are the spectra of produced PD signals

Symmetry of FM signals spectra is inconveneint in musical applications. The issue has been addressed by Palamin et al. [424], who proposed modified FM formu-

las for producing asymmetric spectra, more typical for acoustic instruments. Such spectra may also be obtained using one of the following AdFM techniques [325]: single-sideband AdFM, asymmetric AdFM, or the split-sideband method. The last technique, which is an extension to heterodyne AdFM, is the most flexible among the three.

A modified FM method may be utilised as a source of bandlimited signals for digital subtractive synthesis. The implementation presented by Timoney et al. is based on the following modified FM formula [552]

$$u(t) = e^{(I\cos(2\pi ft)-I)}\cos(2\pi ft) \tag{3.25}$$

where I is the modulation index, and f is the single frequency. The formula yields quasi-bandlimited pulses (Fig. 3.23). Pulses may be used to build other signals. A sawtooth is produced by integrating pulses and centring the result through DC blocking filter. For a square waveform (3.25) is modified by doubling the frequency in exponent, which results in bipolar pulses. The waveform is produced by integrating pulses. Further integrating yields a triangle. Timoney et al. studied the behaviour of such generators and observed, that in case of $f_0 \approx 262$ Hz produced spectrum begins to significantly differ from the ideal digital signal only around 20th harmonic.

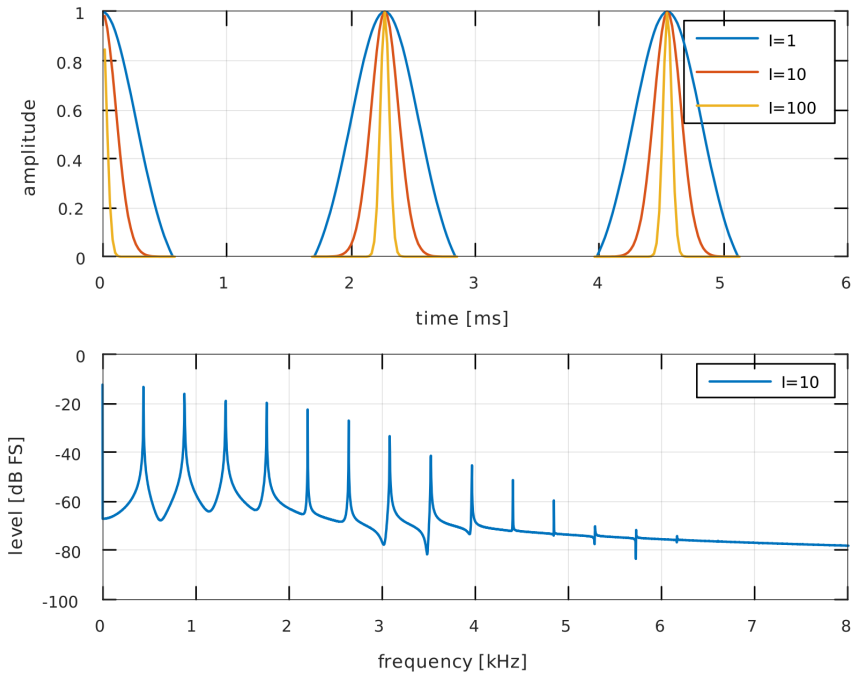


Figure 3.23. Bandlimited pulse signal with 440 Hz frequency, produced by (3.25) for three values of modulation index I (top), and its spectrum for $I = 10$ (bottom)

3.1.2. Waveshaping

When the efficiency in producing simple to handle time-varying spectra is a key objective, waveshaping synthesis may be a solution. The synthesis process involves applying a distortion function to a sound signal. If the function is non-linear, the output signal obtains additional harmonic components. Since FM synthesis shares a similar principle, both are sometimes referred to as distortion synthesis methods.

While in analogue audio nonlinear distortions, if not applied on purpose, might by considered a negative phenomena, there was some initial work by Schaeffer [492] indicating ways to utilise the effect for possible musical purposes. First digital applications of the waveshaping synthesis were presented by Arfib [18] and LeBrun [328].

The most basic digital implementation of the principle consists of a sine oscillator, an amplitude envelope generator, and a *shaping function*, also referred to as *transfer function*, in a form of data array (Fig. 3.24). Signal from the oscillator has its amplitude altered by the EG, and the resulting signal value is used as an index to read output signal value from a waveshaping array. Essentially, the process is limited to multiplication and table lookups, which makes it extremely efficient.

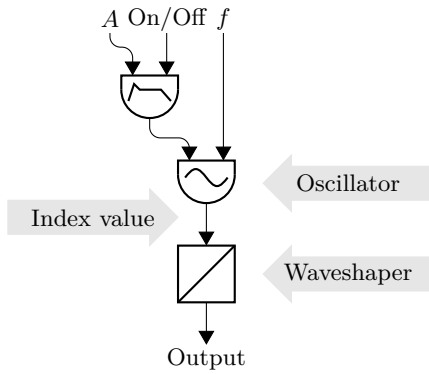


Figure 3.24. Simple waveshaping instrument with key elements indicated

In general, the waveshaping synthesis may be expressed as [155]

$$u_{out}(t) = w\left(u_{in}\left(t\right)\right) \tag{3.26}$$

where w is the shaping function. If the amplitude of the input signal is controller through an envelope $\alpha(t)$, (3.26) changes to

$$u_{out}(t) = w\left(\alpha\left(t\right) u_{in}\left(t\right)\right) \tag{3.27}$$

In practical applications an initial oscillator is not required to produce sine signal – any signal may be used. In order not to exceed indices of array that stores values of shaping function, it is commonly assumed that the signal has to be limited to a range of $[-1, 1]$. A shaping function maps input values $u_{in}[n]$ to output values $u_{out}[n]$ within the same range (Fig. 3.25). While it is not a part of waveshaping

process, the output signal is often further processed using elements of other synthesis methods, such as modulation.

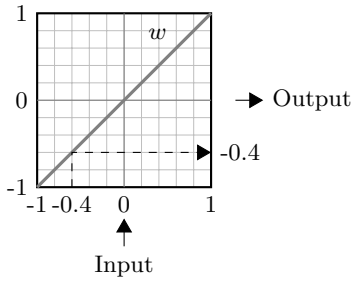


Figure 3.25. A principle of shaping function; here the shaping function w (thick gray line) maps its input to output without changes ($u_{out} = u_{in}$)
Source: author's elaboration, based on Roads [470]

3.1.2.1. Shaping functions

Virtually any function may serve as a shaping function (Fig. 3.26), as well as any data, assuming that it can be limited and stored in a one-dimensional array. In a trivial example a shaping function may simply pass input signal as output (Fig. 3.26a). It can as well apply various effects to the input signal, such as inversion (Fig. 3.26b), attenuation (Fig. 3.26c), or clipping (Fig. 3.26e,f).

Adjusting amplitude of the input signal makes use of various parts of a properly designed shaping function, allowing a sound timbre to be controlled in turn. For instance, one might simulate a sensitivity to amplitude characteristic for acoustic instruments, where distortions start to appear with larger amplitudes, by applying a shaping function that is linear around zero, and distorted towards ends, such as one in Figure 3.26h. Furthermore, an amplitude envelope applied to the input signal results in time-varying spectra. Finally, even more flexibility may be attained by adding a variable offset to the input signal, thus utilising separate parts of a shaping function that do not have to be placed around zero [580].

It is possible to obtain an arbitrary harmonic spectrum by designing a shaping function as a linear combination of the Chebyshev polynomials of the first kind [18, 328]. The polynomials (Fig. 3.27) are defined by the following recurrence relation

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_{k+1}(x) &= 2xT_k(x) - T_{k-1}(x) \end{aligned} \tag{3.28}$$

and have a convenient property

$$T_k(\cos\theta) = \cos(k\theta) \tag{3.29}$$

Therefore, if T_k is used as a shaping function for a cosine input signal that has a frequency f, it produces a cosine output with frequency $k \times f$, or k-th harmonic of the input signal.

An arbitrary linear combination of Chebyshev polynomials

$$w(u) = \sum_{k=1}^{K} a_k T_k(u_{in}) \tag{3.30}$$

where K is the stipulated number of harmonics, and a_k is the coefficient of the polynomial T_k within the combination, used as a shaping function produces a spectrum with the amplitudes of harmonics defined by coefficients a_k. Since only a required number of polynomials is used to design a shaping function, the resulting signal is bandlimited.

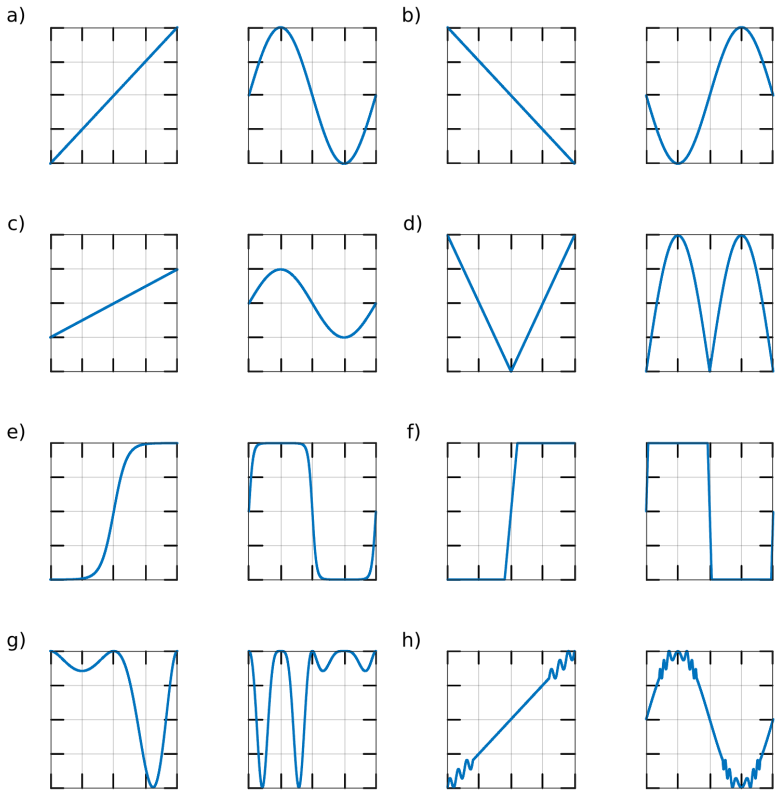


Figure 3.26. Examples of shaping functions; each pair of plots depicts a shaping function (left) and output signal it produces (right) out of a single period of sine as input; input values are presented on x-axis, output on y-axis, both in range of $[-1, 1]$; the functions include: a) identity; b) inversion; c) attenuation; d) scaled absolute value; e) expansion of low signal levels and clipping of high levels; f) clipping; g) combined from several functions; h) amplitude-sensitive distortions (distorted only at higher amplitudes)
Source: author's elaboration, based on Roads [470] and Tolonen et al. [557]

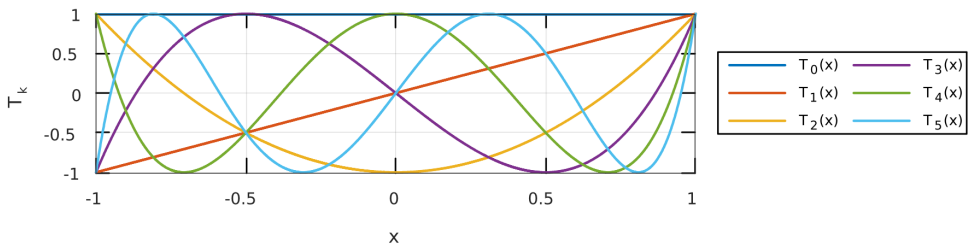


Figure 3.27. First six Chebyshev polynomials of the first kind

3.1.2.2. Amplitude Control

Apart from the simplest cases of shaping functions, such as presented in Figures 3.26a–c, the amplitude of input signal does not define output signal amplitude in a direct way, but controls its spectrum instead. It has some impact on the output amplitude due to various parts of shaping function being utilised, though the relation between input and output amplitude may be complicated, and the output amplitude may strongly vary with changing spectrum.

In order to make amplitude and spectrum control independent, it is convenient to normalise output signal, so that additional amplitude control may be applied in a further stage of processing. Normalisation may be loudness, power, or peak-based, with the latter being the simplest to implement [470]. In peak normalisation another data array is utilised to store amplification coefficients for every possible value of input signal amplitude, i.e. for all values that may be produced by input signal amplitude envelope. These coefficients are applied to the output signal (Fig. 3.28).

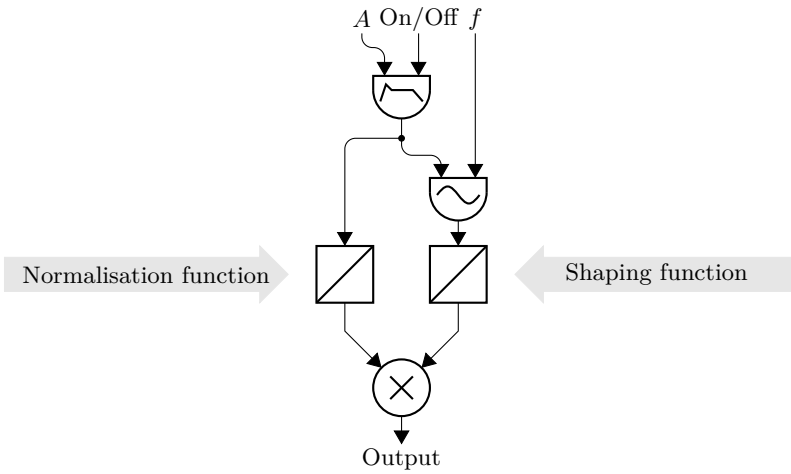


Figure 3.28. Waveshaping instrument with peak normalisation
Source: author's elaboration, based on Roads [470]

3.1.2.3. Variants of Waveshaping

Being a simple and efficient method, waveshaping synthesis can also be relatively easily modified or expanded. The most straightforward way is to apply some kind of postprocessing to the output signal, such as a filter or a modulator [470]. De Poli proposed a variant referred to as **frequency-dependent waveshaping** [155] which through a form of filtering allows to separately control phases and amplitudes of harmonics. In a different attempt Beauchamp applied a highpass filter [43] which allowed him to synthesize brass-like sounds by mimicking damping behaviour of brass pipes.

Modifications can also be applied to the input signal, before the shaping function. It had been already attempted in initial works, where either the sum of several cosines or a frequency modulated signal were applied to produce inharmonic spectra or formant structures [18]. In a different approach, input may be a sampled sound. It produces the effect of a hybrid between the original and synthetic sound.

Linear combination of Chebyshev polynomials is a convenient shaping function if a well-defind static harmonic spectrum is to be produced. Though it is only one possibility, and in an extreme case a shaping function can be even drawn by hand [91]. In **fractional waveshaping** Da Poli experimented with a rational shaping function [155] that can produce spectral envelopes that are exponential or similar to damped cosine, which results in a formant-like structure. The function can also vary over time. It can be efficiently implemented by storing larger array and using its sections that start at addresses indicated by a changing pointer [470].

Redundancy that is the effect of a sine function symmetry can be exploited to further vary output signal by applying separate transfer functions for each quarter-cycle of the input signal, as shown in Figure 3.29. Thus the symmetry of input signal is not transferred to output signal. Such technique is referred to as **quarter-cycle waveshaping** [485] and allows to produce more varied output waveforms out of the same basic sine input.

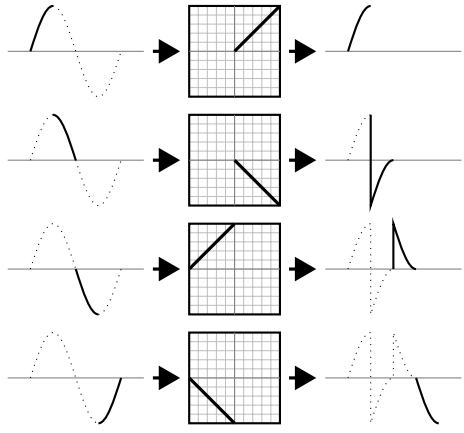


Figure 3.29. Quarter-cycle waveshaping; in each quarter of the input signal cycle a different shaping function is applied, and the output is concatenated
Source: author's elaboration, based on Russ [485]

Freed applied complex arithmetic to formulate a generalised approach to waveshaping analysis and implementation [194]. While it may not be a variant of method, it provides some advantages, e.g. ability to control separate harmonics without constructing the Chebyshev expansion. Finally, a simple arrangement of input signal and shaping function may be expanded by applying a number of such subsets to simulate spectra of existing instrument sounds [44] in a form of resynthesis. A recurrent structure may be designed, where the first subset provides the first approximation of the signal, which is then subtracted from the original spectrum to yield a *residual*. The second subset repeats the procedure using residual as a signal. The scheme can be iterated to refine the result.

3.1.3. Non-Standard Methods

The common assumption regarding sound synthesis is its goal to be mimicking selected aspects of traditional instruments – either their sounds, or their behaviour and usage. However, it is not always the case. A group of methods, referred to as non-standard synthesis methods breaks these traditional bonds. Instead, according to Holtzman, they consist of instructions describing a sound, without reference to a superordinate model [243, 179]. Standard methods, in contrast, are deemed to describe a sound using an acoustic model which is simulated during the process. Roads classifies synthesis methods as non-standard if they are designed to produce new electronic sounds, while standard methods either simulate traditional instruments, or are based on similar principles [470].

3.1.3.1. Waveform Segment

In waveform segment synthesis individual signal samples are connected into larger sound structures of various levels: waveforms, sections, or even entire musical pieces. The process is carried out entirely in a time domain. Spectral structures arise as a result of temporal structures and operations performed in time domain.

Waveform interpolation may be utilises as a synthesis technique. Various interpolation methods can be applied, such as nearest neighbour, also referred to as piecewise constant, linear, exponential, logarithmic, half-cosine, or polynomial, in order to connect endpoints and breakpoints [470], Each of these points is defined by a pair of coordinates. A new waveform may be produced from two source waveforms by their weighted interpolation. The weight may be controlled through an envelope. Spectral properties of interpolated waveforms, including non-uniform interpolation, were studied by Bernstein and Cooper [57]. Mitsuhashi presented a way to control amplitudes of $\frac{n}{2}$ harmonics by altering ordinates of n breakpoints [379].

Brün designed the SAWDUST system that manipulates amplitude points while creating a hierarchy of waveforms, up to a whole musical piece [72]. The elements are manipulated using operations such as concatenation, looping, mixing, and varying.

Another interactive system able to manipulate waveforms and organise them in hierarchies is the SSP designed by Koenig and implemented by Berg [52]. It is based

on serial music selection principles [304, 305] such as alea, series, ratio, tendency, sequence, and group [470].

The direction chosen in SAWDUST and SSP has been explored further in the **instruction synthesis** [467]. In this technique the data loses all references to sound or acoustics, and is simply a sequence of abstract binary values. Operations performed on the data are not specifically oriented towards signal processing, but are purely logical computer instructions. In some early implementations, such as PILE [53], they are modelled on the assembly programming language [488], i.e. they cover very basic, low level operations such as addition, bitwise conjunction and disjunction, loop, or branch. It is also possible to produce random values. In the end, values are treated as signal samples and the whole sequence is considered an output signal. Even though in PILE instructions are executed in a virtual machine, the process is very efficient due to a character of data and instructions. A different approach was proposed by Holtzmann [244], where a user worked with a high level program generator, which in turn produced short synthesis programs.

3.1.3.2. Graphics Synthesis

The idea behind graphics synthesis is to generate acoustic signal on the basis of a picture, or convert a sound into a picture, edit it as such, and convert it back to sound [470]. First attempts at such technique date as early as 1920s [462]. In case of digital sound initial experiments with controlling it using graphics were carried out by Mathews and Rosler [357].

Since there is no universal principle regarding the conversion between a sound and a picture, the technique is widely open to experiments. Among many possible approaches, graphical data may control various aspects and parameters of generated signal, and can utilise different sound production mechanisms. Actually, the graphics synthesis may be considered more a control interface for a synthesizer than the synthesis engine itself. However, the results often demonstrate distinct qualities, particularly with hand-drawn images. Not unlike granular synthesis, graphics synthesis operates on a level different than most of other synthesis methods. Instead of generating separate sound events representing notes, it is often utilised to produce larger sound structures, and even whole musical pieces.

Early image-related techniques implemented in analogue instruments were based on photoelectric signal generators. They utilised a transparent medium, such as an optical disc or a film, containing a *phonogram*, i.e. an image of a waveform [470] which formed a 'track' with varying transparency. The conversion involved applying a light source on one side, and a light detector on the other side of a medium (Fig. 3.30). Depending on varying opacity while the medium was shifting or rotating in relation to the light source, various amount of light was getting to the detector which in turn produced a signal. The actual implementations varied, but a general principle has been applied in a large number of commercial instruments, from Cellulophone and Superpiano in 1920s, through Welte Light-Tone Organ in 1930s, to Optigan [81] and Orchestron in 1970s. In 1940s the method was used by McLaren, who drew waveforms directly on the optical soundtrack to produce synthetic sound for films [361].

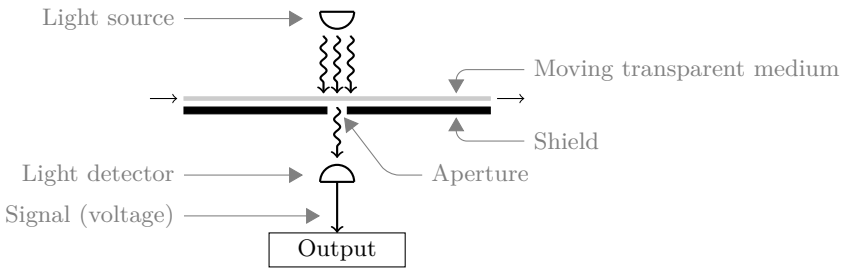


Figure 3.30. Conversion of an audio signal represented by a one-dimensional image on a transparent medium; movement of the medium changes transparency of its part positioned over the aperture, resulting in variations of light stream reaching a light detector, which in turn produces an electric signal

The next step in graphics synthesis was to move from a simple waveform-like sound representation to a more abstract, parametric approach. Oramics Graphic System designed by Oram in 1957 [170, 415] allowed to draw control functions for an analogue synthesizer, including frequency, amplitude, and filter settings, as well as magnitude of *vibrato* and *tremolo* effects. Even though Oramics shared basic technique with the approach used by McLaren, having ten synchronised strips of a transparent sprocketed film and its rich control abilities, it may be considered a kind of fairly advanced sequencer. Other synthesizers allowed different forms of medium to be utilised, such as paper and conductive ink [462]. In 1979 a rudimentary graphics synthesis has been implemented in a digital commercial synthesizer Fairlight CMI Series I, where a user was able to draw waveforms by hand on a computer screen using a lightpen [485]. However, graphical control data was already evolving into forms resembling a sonogram [470], and basic approach with one-dimensional control functions has been gradually abandoned.

Early, simpler forms of graphics synthesis, based on transparent media and converters similar to that shown in Figure 3.30 applied a direct approach of converting light intensity to some immediate synthesis parameter, such as frequency or amplitude. Further development, with a notable example of Oramics, led to 'reversing' the process of displaying a control signal on cathode ray tube display of an oscilloscope, allowing to draw various shapes of control functions on films. Yet, such association of sound and image is still a very basic one. Advancements are possible through implementation of digital technique with computers, high resolution colour displays, graphics tablets, and – more recently – small computing devices equipped with touchscreens, serving simultaneously as both, graphical input and output device. Computers allow to introduce complex middle layers of interpretation, which in turn can be exploited to experiment with approaches much more flexible than simple graphical representations of control functions.

A general discussion over the relation between a graphics and a sound, or more specifically between graphical form and auditory event, has been given by O'Sullivan [422], who also developed a controller for the graphical sound synthesis

aimed at exploring cross-modal perceptual analogies. He seeks the relationship universal enough not to require grounding in music theory or cultural conventions to understand. Some clues as to general principles of such relationship may be found in the phenomenon of synaesthesia, and colour-mapping problems that resulted from experiments carried out by Scriabin who designed the *keyboard of light*. O'Sullivan mentions brightness, size, and some shape-related characteristics as image quantities more agreeably assigned to particular sound qualities. While designing cross-modal mappings, such as auditory interpretation of an image, three strategies may be considered [481]. *One-to-one* is the simplest, the least expensive, and intuitive, but less expressive. *Divergent*, where one input controls several outputs, is less intuitive in controlling sound, while *convergent* mapping requires some training, but may be more expressive than *one-to-one* approach. Even though these strategies were initially aimed at gesture control, they can be applied in graphics synthesis as well, through associations between gestures and shapes. As of sound and image features, O'Sullivan experimented with simple relations, such as coordinate or distance to frequency, size to amplitude, as well as with more complex, such as line curvature to frequency modulation index, or irregularity to inharmonicity.

Experiments with graphical control over synthesis process, such as described by O'Sullivan, grant insights into issues of association between image and sound, yet they are still rooted in a traditional understanding of a controller, which is simply a means to produce events representing notes. A move from analogue to digital made graphics synthesis more flexible and allowed it to evolve from a simple synthesizer controller to a composition environment. Various systems have been developed, such as the UPIC (Unité Polyagogique Informatique de CEMAMu) created at the Centre d'Etudes de Mathématique et Automatique Musicales (CEMAMu) with participation of Xenakis [611]. UPIC utilises a large graphics tablet connected to a computer with a vector display. A sound is produced at two levels [470]. Firstly its microstructure is designed by drawing waveforms and envelopes, with computer able to perform interpolation between manually entered points. Secondly, a time-frequency structure, referred to as a score page, is created by drawing lines, or arcs. Data processing characteristic for graphical data may be applied, such as moving or stretching of individual elements, copying, cutting, and pasting. In early versions drawing on the tabled produced the result on screen immediately, but generation of a sound signal took considerable amount of time. Due to usage of faster computer, later improvements allowed to play edited page in real-time, thus making UPIC a performance instrument [354].

3.1.3.3. Motion-Driven Synthesis

Graphics control applied to sound synthesis leaves a distinct footprint. Particularly some features of hand-drawn images, such as certain kinds of irregularity, may lead to audible and characteristic effects. In a similar manner sound synthesis may be driven by motion data. The data can originate from video, but other sources are becoming increasingly more accessible, bringing different kinds of motion sensors, i.e. infra-red based, or relying on accelerometers.

Large amount of work has been carried out in the area of video-related motion-driven synthesis, geared towards application in computer animation, video games, or virtual-reality [100]. Even though most of the research work aims at synthesising sound effects and ambient sounds, there is also a number of musical implementations. [226, 376].

Mishra and Hahn [376] proposed a methodology to map sounds to motions, and a system that applies this methodology to produce soundtracks. Motion control system supplies the data which is filtered, and mapped to musical constructs, producing either a digital score or a real-time performance. Figure 3.31 presents how sound and image production pipelines are bound to generate synchronised visual and aural output. Mapping is carried out at two levels: local, or intra-stream, and global, inter-stream. The former is applied to internal sound structures. The latter binds streams together by attaching global constructs. Depending on sound structures that are to be utilised different methods may be applied, such as *timbre-trees* [226] or musical constructs.

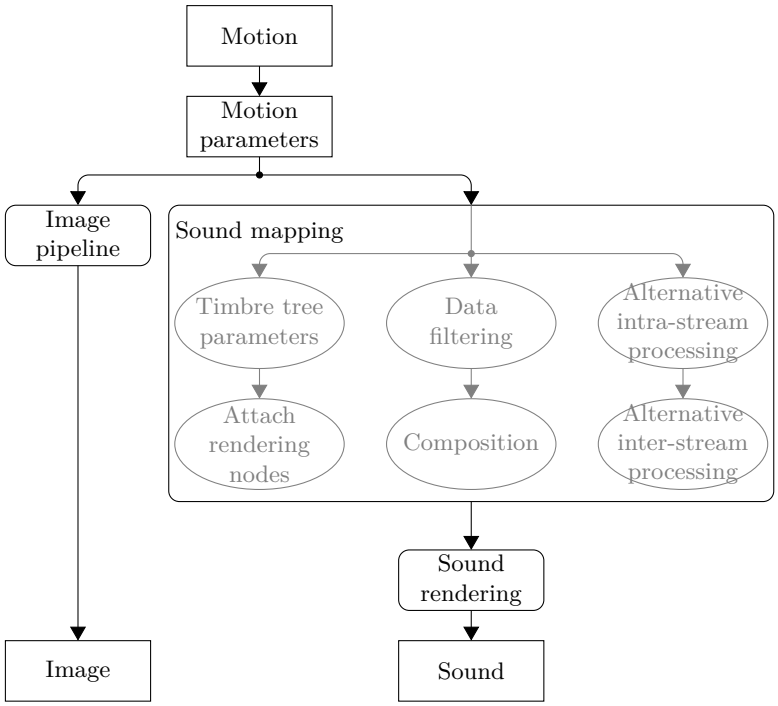


Figure 3.31. Integration of sound and motion; the first stage of sound mapping is applied on a level of internal sound structures, the second is responsible for binding streams together
Source: author's elaboration, based on Mishra and Hahn [376]

Motion parameters vary depending on application. Position and orientation, as well as joint angles can be obtained through kinematic techniques, such as *keyframing*. If a motion is based on physical simulations additional data may be acquired, such as

response to collision events. Virtual reality sets may also add data regarding gesture or locomotion. Mishra and Hahn propose [376] that if the music produced is to consist of a number of parts, some may be driven by a direct position, some by computed velocities, and others by acceleration. Filters are applied to produce required number of parts from available number of data streams.

When motion is used to drive production of sound effects, mapping motion parameters to sound parameters should correspond with real physical interaction. Though when the system is applied to producing music, the mapping is abstract. Depending on exact sound production method various parameters describing internal sound structure may be available. As of generalised constructs usual music-related data can be controlled, including pitch and amplitude, as well as effects-related parameters, such as pitch-bend, delay, or reverberation.

3.1.3.4. Noise Modulation

It is usual for distortion synthesis to utilise periodic waveforms, e.g. a sine, or quasi-periodic, such as sampled instrument sounds. However, it is also possible to modulate a periodic waveform using a filtered noise in a technique referred to as the noise modulation. Such approach is possible in amplitude and frequency modulation as well as in waveshaping synthesis.

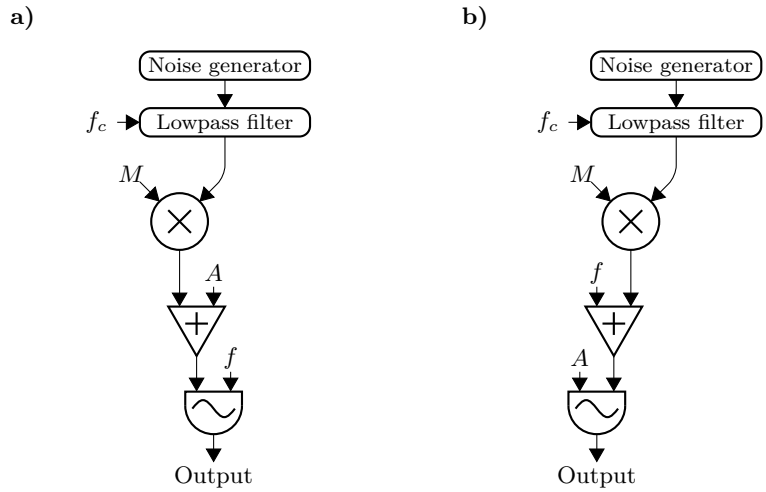


Figure 3.32. Noise modulation instruments in AM (a) and FM (b) configuration; parameter M controls the depth of modulation, and f_c is the filter cut-off frequency
Source: author's elaboration, based on Roads [470]

In digital implementations noise is produced as a series of random numbers by random number generator (RNG). Earlier computer systems were only able to generate random numbers through computational algorithms. Results of such algorithms are sometimes referred to as *pseudo-random numbers*. The algorithms produce periodic sequences of numbers. Pseudo-random sequence should have a very long period, so

that the periodicity is not manifested in produced data. Moreover, it should have no apparent internal pattern and, depending on projected application, shall pass at least some statistical tests for randomness [597]. One of the most common pseudo-random number generators (PRNG) is the **linear congruential generator** (LCG) [301, 203], where

$$X_{n+1} = (aX_n + c) \bmod m \tag{3.31}$$

where m is the modulus, a is the multiplier, and c is the increment. All three values are integer numbers, and $m > 0$, $0 < a < m$, $0 \leq c < m$.

The formula is recursive, and in order to begin generation of random numbers some initial value of X_0, referred to as *seed*, has to be chosen. Each time a generator is started and different random sequence is required, a different seed has to be provided. There may be a problem with source of new, random seeds in deterministic system. Various computer systems solve it differently, e.g. by providing a virtual device that gathers environmental noise from device drivers and other sources into an entropy pool. A crude solution is to base seed on selected bits of current system time, read with possibly high precision.

Selection of modulus, multiplier, and increment determines qualities of LCG, e.g. length of its period. Value $m-1$ determines the maximal length of produced sequence, but it is only attainable under certain circumstances. The Hull–Dobell Theorem [257] states, that LCG has a full period for all seed values if and only if:

- m and c are relatively prime[5],
- $a - 1$ is divisible by all prime factors of m,
- $a - 1$ is divisible by 4 if m is divisible by 4.

Particular compilers utilise different values of m, a, and c (Tab. 3.3). Some ready to use sets of parameters for multiplicative LCGs, which are the subset of LCGs, of different sizes and good performance with respect to the spectral test are presented by L'Ecuyer [329]. LCGs produce numbers which plotted in two or more dimensions form lines or hyperplanes. Spectral test compares the distance between planes, which in good generators should be small [301].

Apart from PRNGs random numbers may be produced using physical methods. Some computer systems may allow to source random values from e.g. a thermal noise, a radio noise, or a clock drift. Due to better efficiency of PRNGs than RNGs based on physical phenomena, in speed-constrained applications the latter are good seed sources for algorithmic PRNGs such as LCGs. In musical applications chaotic noise can also be produced by some synthesis methods, particularly granular and modulation synthesis. Generally, for the purpose of sound synthesis statistically acceptable white noise may not always be the best choice. In some cases alternative methods can be considered. Such methods may produce distributions of random numbers different then usual uniform, or even introduce some forms of complex internal patterns, which would be regarded as undesired in other applications of RNGs.

In noise modulation lowpass-filtered signal from PRNG controls the amplitude or frequency of the oscillator (Fig. 3.32). The result depends largely on the filter

[5]Two numbers are relatively prime if the only positive integer that divides both of them is 1.

cut-off frequency (Fig. 3.33). For values below 20 Hz the effect may be considered an *aleatoric tremolo* or *aleatoric vibrato*. If cut-off frequency falls within an acoustic range the effect of modulation is a band of noise centred around carrier frequency.

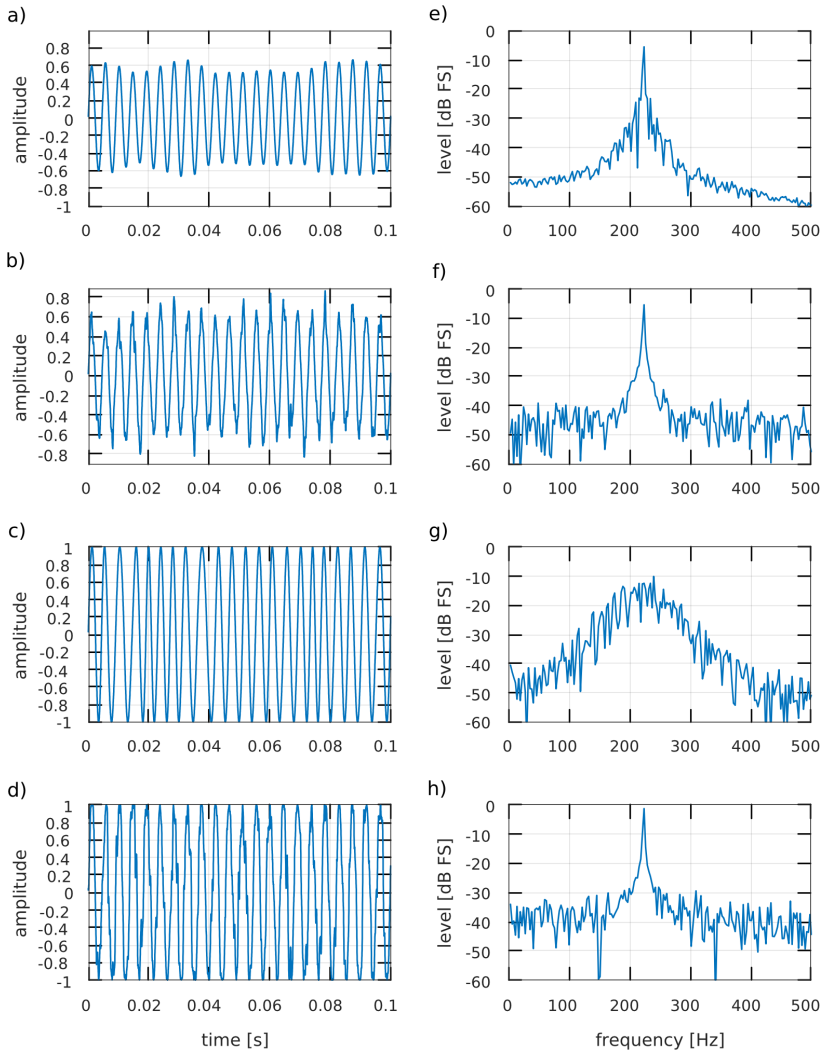


Figure 3.33. Effect of a 220 Hz sine carrier modulated using filtered noise with various cut-off frequency value (f_c): a) waveform of amplitude-modulated signal with $f_c = 15$ Hz; b) AM with $f_c = 1000$ Hz; c) FM with $f_c = 15$ Hz; d) FM with $f_c = 1000$ Hz; the remaining plots (e–h) present respective spectra

In the waveshaping synthesis noise may be added to a shaping function to change the instantaneous signal amplitude. In the simplest case a varying amount of white noise can be added to a shaping function (Fig. 3.34a-b). Like in noise modulation

the noise may be filtered beforehand (Fig. 3.34c). Amount of noise may also depend on the input signal amplitude (Fig. 3.34d), which allows to control a content of noise in the output signal.

Table 3.3. Examples of parameter values commonly used in linear congruential generators (3.31)

Used in	m	a	c
ANSI C	2^{31}	1103515245	12345
GCC/glibc	$2^{31}-1$	1103515245	12345
Borland C/C++	2^{32}	22695477	1
Java and rand48 in POSIX and glibc	2^{48}	25214903917	11

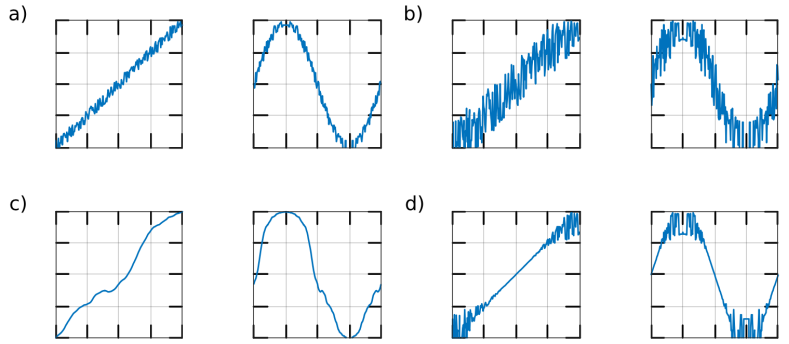


Figure 3.34. Waveshaping functions with random component; each pair of plots is a shaping function (left side) and output signal it produces (right) from a single period of a sine as input; in each case the linear shaping function served as a base, and the noise component was: uniform with 10% amplitude (a), uniform with 40% amplitude (b), uniform and low-pass filtered with 20% amplitude (c), increasing with higher amplitudes (d)

3.1.3.5. Stochastic Waveform Synthesis

Musical applications of stochastic techniques have a long history. Most are centred around composition [340, 274, 15, 16, 611], but there is a number of works that describe application of probability distributions, fractals, and other stochastic techniques in sound synthesis [598, 509, 611, 470].

Majority of sound synthesis techniques attempt to design complex sounds through modification and combination of simple elements, such as periodic functions. According to Xenakis [611] it may be considered as adding 'disorder', and there is an alternative way: to start with random, disordered elements, and add 'order' in a form of constraints. Xenakis proposed a set of strategies regarding dynamic stochastic waveform generation [611, 470]. As a starting point, probability distributions may be used directly to create waveforms. Probability functions may be multiplied by themselves or combined through addition. Samples can be produced by treating amplitude and

time as random variables that may bounce between elastic boundaries. There may be a hierarchy or chain of probability functions, where results produced by one function are passed to another, where the last function in the chain produces samples. Finally, such hierarchy may also assume forms more complex than simple chains.

Stochastic waveform synthesis is applied, and combined with interpolation synthesis, in the GENDY program [509]. The program works with a waveform represented as a polygon that is bounded between amplitude and time boundaries. Signal samples are calculated by linear interpolation between polygon vertices. Vertices are calculated according to various stochastic distributions, and subsequent waveforms are stochastic variations on previous ones – vertices are repositioned. If during this variation a vertex would cross a boundary, a mirror condition is applied, and it is reflected back. Such constraints have an effect on sound timbre if amplitude boundaries are applied, due to introduction of discontinuity into a waveform, and pitch effect in case of time boundaries. Multiple mirror conditions applied together allow to produce *vibrato* and *tremolo* effects.

3.1.3.6. Cellular Automata Synthesis

Musical sounds often manifest various forms of patterns, either in spectral or temporal representation. Patterns can be observed in structures on various levels, from elements of a single sound, to whole musical pieces. Therefore it was only natural to attempt at applying cellular automata (CA), known for ability to produce complex structures, as a means to experiment with creating new sounds [371], as well as organising sounds into larger musical structures [58, 368, 370, 372, 240].

CA were initially intended as a tool for computational modelling geared towards studying self-reproducing and self-organising mechanisms [120]. They are dynamic systems, implemented as arrays of elements, usually one or two-dimensional, although higher number of dimensions may also be encountered. A single element is referred to as *cell*. Cells are subject to *transition rules* that determine future state of a given cell considering states of its neighbouring cells. All cells are updated simultaneously, so that a whole array evolves in parallel steps. Hence CA may be regarded as existing in a discrete space, represented by an array, and in a discrete time, measured in subsequent steps of evolution. As such, they can be conveniently applied in a digital sound synthesis.

In more sophisticated implementations cells can represent some complex processing units, however it is common that a state of a cell is represented simply by an integer number. Class of such cells is referred to as p-state CA, since their possible states are $0, 1, 2, \ldots, p-1$ [374].

An initial state of cells within an array may be set manually, randomly, or according to some assumed rules. In subsequent time steps the arrangement of cells in an array evolves, and various patterns emerge, depending on initial arrangement and transition rules. In order to observe this behaviour an array is often presented as a graphical map, where cell states are represented by indexed colours. An example set of transition rules for binary[6], one-dimensional, nearest neighbour CA, which is

[6]In this case 'binary' means a two-state cell that can assume either value of 0 or 1.

the simplest case of CA, is presented in Figure 3.35. When cell arrangement is two-dimensional and CA is of nearest neighbour type, usually either four or eight adjacent cells are regarded as neighbours. The former is referred to as von Neumann neighbourhood, and the latter – Moore neighbourhood (Fig. 3.36). It is possible to consider not only nearest cells, but extended neighbourhood as well, which is controlled by *range* parameter r. In case of the nearest neighbour CA $r = 1$.

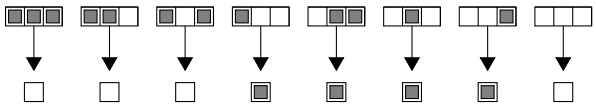


Figure 3.35. An example of a complete set of transition rules for a binary, nearest neighbour one-dimensional cellular automaton; filled cell represents state 1, and empty – state 0
Source: author's elaboration, based on Miranda [374]

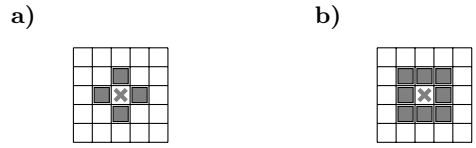


Figure 3.36. Types of two-dimensional neighbourhood: a) von Neumann; b) Moore; nearest neighbours of crossed cell are represented in gray

The array is usually finite. In order to avoid edge problems either border cells have constant states or the array is considered to have a toroidal arrangement, also referred to as periodic boundary conditions, i.e. top and bottom cells are neighbours, as well as rightmost and leftmost cells. In a two-dimensional periodic array of size $X \times Y$ von Neumann nearest neighbours of cell (0,0) are: (1,0), (0,1), ((X−1),0), and (0,(Y−1)).

For the purposes of sound synthesis one-dimensional CA are particularly important due to a straightforward way they can be interpreted – as an evolving waveform or spectrum. Wolfram extensively studied their properties [609]. In case of the nearest neighbours CA eight different arrangements of a cell and its nearest neighbours (Fig. 3.35) lead to 256 possible transition rules. More generally, a total number of possible automaton rules is expressed as

$$R_N = p^{p^{2r+1}} \tag{3.32}$$

where p is the number of cell states, and r is the range. Each set may be assigned with a code

$$C_f = \sum_{n=0}^{(2r+1)(p-1)} p^n f(n) \tag{3.33}$$

where f is the single argument function that specifies the cellular automaton rule

$$a_i^{(t)} = f\left(\sum_{j=-r}^{j=r} \alpha_j a_{i+j}^{t-1}\right) \tag{3.34}$$

where $a_i^{(t)}$ and $a_i^{(t-1)}$ are current and previous cell states, and α_j are integer constants. If all constants $\alpha_j = 1$, the state of a cell depends on the total of its neighbours only, while their particular configuration is irrelevant. Such rules are referred to as *totalistic*. Figure 3.37 presents examples of evolution resulting from applying rules presented in Figure 3.35 to a various initial array arrangements.

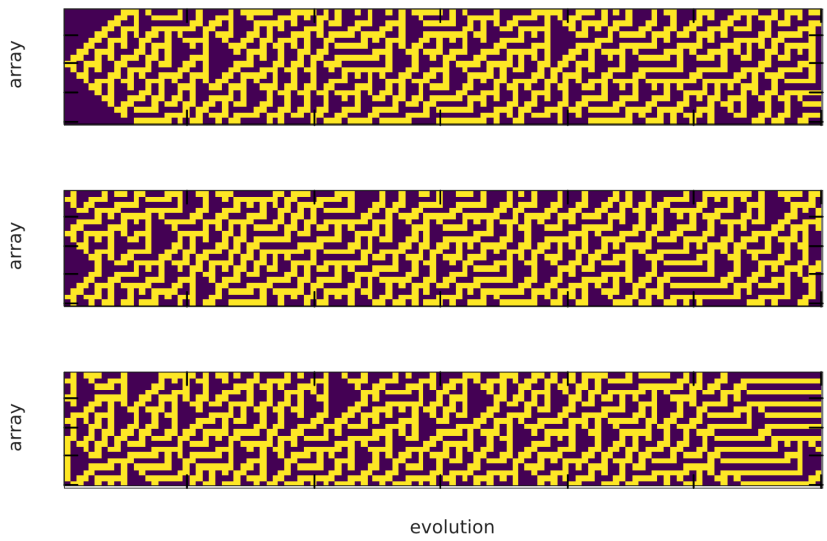


Figure 3.37. Examples of a one-dimensional, nearest neighbour binary CA with transition rules given in Figure 3.35 applied to three different initial array arrangements; the horizontal axis represents a series of subsequent time steps of the evolving array located on the vertical axis; top example begins with a single cell, middle and bottom – with two different random arrangements; after some time all three cases produce a similar pattern

Wolfram proposed to divide CA into four classes, according to their behaviour (Tab. 3.4). Langton studied the connection between computation and phase transition using CA [316]. He introduced parameter λ that allows to predict general properties of a pattern produced by CA and classify it as fixed, cyclic, complex, or chaotic. In sound synthesis λ allows to estimate auditory result of particular CA rules.

Miranda [374] mentions three examples of CA that have been applied in sound synthesis: Game of Life, Crystalline Growths, and Chemical Oscillator. The basic variant of Game of Life, invented by John Conway, is a binary, two-dimensional, nearest neighbour CA with Moore type of neighbourhood [55]. State 1 is referred to

as a 'live' cell, and 0 as a 'dead' one. The original rules are presented in Table 3.5. They are sometimes abbreviated as 'B3/S23', where 'B' symbolises number of alive neighbours required for birth, and 'S' stands for number or numbers of alive neighbours required for survival. Other numbers result in death of a cell. Game of Life allows to observe various patterns that can expand, freeze, die out, change shape, oscillate, travel, or produce another patterns. Interesting patterns are given names, such as *glider* or *lightweight spaceship* for travelling structures, *blinker*, *toad*, or *pulsar* for oscillators, *die hard* for a *Methuselah* kind of structure that evolves for a large number of periods, and finally dies out, or *glider guns* for structures that periodically produce gliders.

Table 3.4. Classes of one-dimensional CA according to Wolfram [609]; a code defines rules according to (3.33); class IV automata require either range parameter $r > 1$, or number of states $p > 2$

Class	Evolution result	Codes
I	Homogeneous state	0, 4, 16, 32, 36, 48, 54, 60, 62
II	A set of separated simple stable or periodic structures	8, 24, 40, 56, 58
III	Chaotic pattern	2, 6, 10, 12, 14, 18, 22, 26, 28, 30, 34, 38, 42, 44, 46, 50
IV	Complex localised structures	20, 52

Table 3.5. The original rules for Game of Life CA [55]; future state of a cell depends on its current state and the sum of states of its eight neighbouring cells

Description	Cell state	Sum of neighbours	New cell state
Birth	0	3	1
Death by overcrowding	1	≥ 4	0
Death by exposure	1	≤ 1	0
Survival	1	2 or 3	1

Crystalline Growths is a two-dimensional CA [160]. It is non-binary, i.e. assumes a larger number of states represented by colours. The transition rule is, that a cell in state k dominates its neighbours in state $k-1$. These neighbours assume state k in the next period. A cell that is dominated can at the same time dominate another cells. The rule is periodic, therefore a cell in the lowest state zero dominates cells in the highest state $p-1$. Crystalline Growths evolves into stable, patchwork-type patterns [374].

Gerhardt et al. [204] introduced an automaton for modelling a Belousov–Zhabotinskii chemical reaction. It was adapted by Miranda [372, 374] in the Chemical Oscillator (ChaOs) to produce patterns associated with oscillatory neuronal activity. Cells in ChaOs represent neurons. A cell can be in a quiescent state, also referred to as polarised, in one of n depolarisation states, or in fired state. Cells interact

through a flow of 'electric current'. A state of a cell is characterised by threshold values V_{min} and V_{max}. A cell remains quiescent if its internal potential $V_i < V_{min}$. When $V_{min} \leq V_i < V_{max}$ a cell is being depolarised. Each cell has a capacitor k and two resistors r_1 and r_2. Resistors attempt to maintain $V_i < V_{min}$. If they fail a cell becomes polarised. Polarisation increases in subsequent time steps with a rate controlled by the capacitor. Once $V_i = V_{max}$ a cell is fired and will be restored to quiescent state in the following time step. ChaOs cells are usually arranged in two-dimensional arrays with Moore type of neighbourhood and are updated simultaneously according to the following rules

$$m_{x,y}(t+1) = \begin{cases} \left\lfloor \frac{A}{r_1} \right\rfloor + \left\lfloor \frac{B}{r_2} \right\rfloor & \text{if } m_{x,y}(t) = 0 \\ \left\lfloor \frac{S}{A} \right\rfloor + k & \text{if } 0 < m_{x,y}(t) < p-1 \\ 0 & \text{if } m_{x,y}(t) = p-1 \end{cases} \tag{3.35}$$

where $\lfloor \cdot \rfloor$ is the integer part of the value, $m_{x,y}(t)$ is the state of a cell, x and y are the cell coordinates, A is the number of fired neighbours, B is the number of depolarised neurons, and S is the sum of the states of all the neighbours. ChaOs evolves into oscillatory cycle of patterns.

Several attempts have been made to develop synthesis systems based on CA. One-dimensional binary CA has been applied by Orton et al. [417] in granular synthesis. Kreger [308] modified sound spectra using binary one-dimensional CA to control filter coefficients in resynthesis. Another granular implementation of CA was presented by Vaidhyanathan et al. [566]. More synthesis systems based on CA were presented by Burraston [88].

One of the most straightforward applications of CA in sound synthesis is the Linear Automata Synthesis (LASy) proposed by Chareyron [112]. It may be considered as a variant of evolving table lookup methods, such as the Karplus–Strong algorithm [286]. These methods share a general principle of periodic reproduction of samples stored within a one-dimensional array. Unlike in wavetable method, stored signal samples are subjected to modifications.

Production of a sound event starts with filling the array with initial data, usually random. Once initialised, array is read in cycles to produce sound samples. In each read cycle array is modified through some feedback function (Fig. 3.38). Karplus–Strong uses a lowpass filter, while LASy applies a set of CA rules. Each signal sample represents a single cell in one-dimensional array of automata. Transition rules are applied after each read cycle to modify cell states, thus leading to evolution of the signal produced. In most cases the fastest evolution occurs in earlier stages, while later periods tend to stabilise. However, it depends on the choice of transition rules and assigned class of CA (Tab. 3.4). Particularly chaotic class produce interesting complex sounds with unpredictable spectra. It has to be considered though, that the method has limited control capabilities and prediction of the auditory results is difficult.

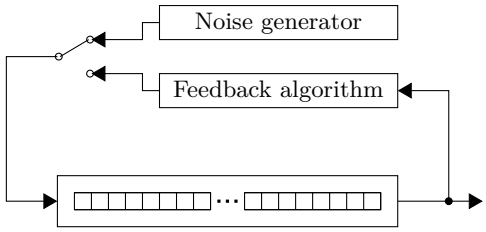


Figure 3.38. In Linear Automata Synthesis CA are utilised as a feedback algorithm to modify samples in a time-varying lookup table
Source: author's elaboration, based on Miranda [374]

While LASy applies CA in a table lookup synthesis, Chaosynth (Fig. 3.39) employs ChaOs automata to control parameters of oscillators producing grains for a granular synthesizer [374]. Parameters controlled by CA are the frequency and the amplitude. Each grain is produced by summing output of several oscillators, able to generate signals such as sine, square, sawtooth, or bandlimited noise. One time step of CA evolution produces a single grain. While typically states of a cell are represented by colours, in Chaosynth they are associated with particular frequency and amplitude values. A single cell state represents both values at the same time. These values are defined by a user.

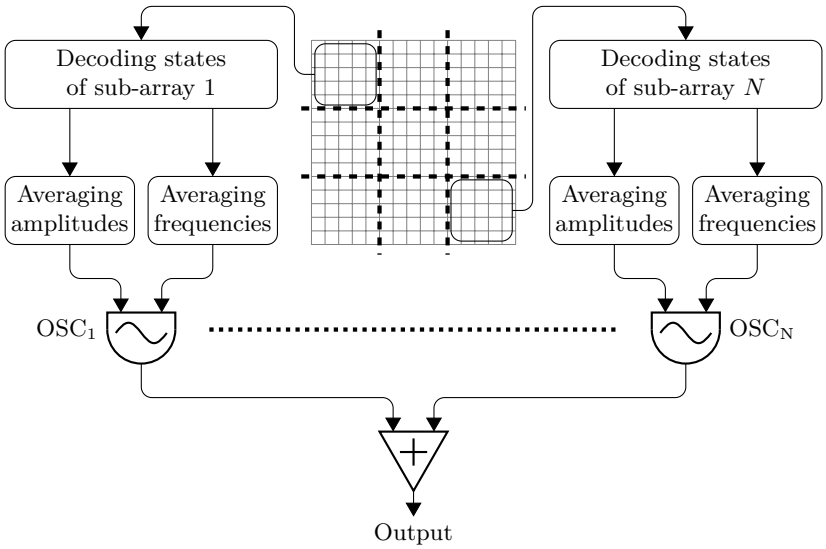


Figure 3.39. Chaosynth produces sonic grains by utilising two-dimensional array of ChaOs CA, divided into N sub-arrays, to control amplitude and frequency of N oscillators; each time step of the automata evolution produces a single grain
Source: author's elaboration, based on Miranda [372]

ChaOs arranges cells in a two-dimensional array. Chaosynth subdivides this array into a number of uniform sub-arrays. Each sub-array controls a single oscillator (Fig. 3.39). Control value is defined as an arithmetic mean of all cell states within a sub-array. Thus even with relatively low number of possible cell states it is possible to obtain much finer parameter resolution by averaging over a larger number of cells with different values. Synthesis process may be controlled either by changing properties of the system, or values of its parameters. Properties are the size of array and sub-arrays, as well as choice of the oscillator signals. Parameters include frequency and amplitude values mapped to cell states, and quantities related to ChaOs CA, controlling transition rates, i.e. r_1, r_2, and k. Even though Chaosynth is more flexible in its control possibilities compared to LASy, it is still difficult to predict and describe auditory effect of its synthesis process. Some attempts have been made at establishing taxonomy for these new categories of sounds [374].

3.1.3.7. Waveset Distortion

The waveset distortion technique, introduced by Trevor Woshart [373], may bear some similarity to granular synthesis. Like granular synthesis it also transforms a sampled sound by the means of fragmentation and rearrangement. However, the concept differs in the fragmentation approach. Waveset distortion divides a sound recording into units referred to as *wavecycles*. A wavecycle is a sample segment between two subsequent zero crossings. Groups of wavecycles are referred to as *wavesets*.

The principle of waveset distortion synthesis relies upon performing particular operations over wavecycles or wavesets [373]. The first group of operations is aimed at montage. It includes addition and removal of wavecycles in a waveset. It is also possible to substitute selected wavecycles with other waveforms or silence periods, as well as interleave wavecycles of two different sounds. The second group involves modulation of either wavecycles, or whole wavesets. Another kind of operation is the application of amplitude envelope on a wavecycle level. The final group are the permutations of wavecycles.

3.1.3.8. Sequential Waveform Composition

Selected principles of the instruction synthesis and the stochastic waveform method were mixed in a technique of sequential waveform composition described by Chandra [111] and Miranda [373]. In this technique signal is assembled in a time domain out of segments grouped into sequences, referred to as states, which are repeated to obtain required sound duration. Pitch is controlled by changing a state duration, while characteristics of timbre depend on features and order of segments within a state.

There are three types of segments, named wiggle, twiggle, and ciggle, as described in Table 3.6 and presented in Figure 3.40. While wiggle and twiggle are simple linear

sequences, the ciggle is curved, thus requiring three points to define, and is given by the following polynomial [111]

$$y(x) = y_1 + (x - x_1)\left(\frac{-y_1 + y_2}{-x_1 + x_2} + \frac{(x - x_2)\left(\frac{y_1 - y_2}{-x_1 + x_2} + \frac{-y_2 + y_3}{-x_2 + x_3}\right)}{-x_1 + x_3}\right) \tag{3.36}$$

where (x_1, y_1) is the start point, (x_2, y_2) is the peak point, and (x_3, y_3) is the end point. If required, a segment may be slanted (Fig. 3.40b) to start with a sample value that ended a previous segment, preventing from discontinuities in signal values. It is possible to introduce new segment types besides the three described by Chandra.

Table 3.6. Types of segments in sequential waveform composition, according to Chandra [111]

Name	Description	Parameters
Wiggle	A stream of signal samples of the same value	Sample value, number of samples
Twiggle	A peak in sample values with linear slopes	Sample value at the base, sample value at the peak, number of samples, peak position
Ciggle	A twiggle with curved slopes	Polynomial parameters x_1, x_2, x_3, y_1, y_2, y_3, as in Eq. 3.36

A state is composed by firstly, defining segments that are to be utilised, and secondly, by specifying their order, with possible multiple occurrences. If, for example, defined segments include two wiggles (w_1 and w_2), and three ciggles (c_1, c_2, and c_3), the following sequence is one of the possible states: [c_3, c_1, c_3, w_1, w_2, c_1, c_2]. If a pitched sound is to be produced, a total state duration needs to have a fixed value, which in the simplest case corresponds to a period of produced signal

$$N_s = \left\lfloor \frac{f_s}{f_0} \right\rceil \tag{3.37}$$

where N_s is the number of samples within a state, f_s is the sampling frequency, f_0 is the stipulated fundamental frequency, and $\lfloor \cdot \rceil$ is the nearest integer.

Sequential waveform composition allows to produce evolving spectra (Fig. 3.40g) by specifying transformations applied to variables from Table 3.6. Transformation of a variable is defined by four values: initial, maximal, minimal, an rate. When a signal is produced a variable is set to its initial value. With each repetition of a state immediate variable value is increased by a rate, up to the point of reaching a maximal value. Thereafter it is reset to a minimal value, and a cycle repeats. Chandra points out [111] that interesting effects may be obtained by altering variables synchronously, or in some special relation, such as following the harmonic series, or being relatively prime.

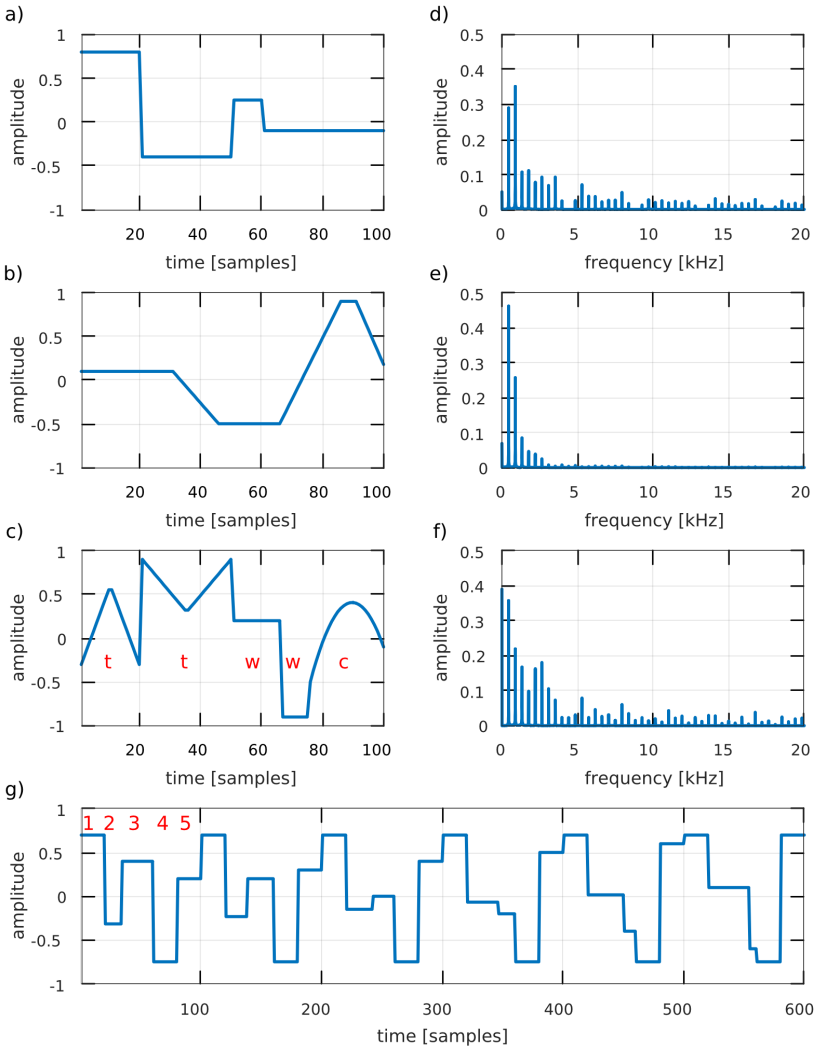


Figure 3.40. Examples of sequential waveform composition; plots a–c present three cases of a single state lasting 100 samples, plots d–f present their respective spectra ($f_s = 44100$ Hz); (a) uses only wiggle segments, (b) makes every second wiggle slanted, while (c) supplements wiggles with two twiggles and a ciggle, as denoted by red characters; the last example (g) presents an evolving signal, with segments 2 and 3 altering their lengths and level, and segment 5 altering level only

3.1.3.9. Neural Audio Synthesis

One of more recent sound synthesis techniques is founded on principles and facilities of **deep learning** methods, that are beginning to find their way into various areas of art-related research, including music creation. The neural audio synthesis,

presented by Engel et al. [182], is a data driven approach to sound synthesis. Instead of relying on either various arrangements of usual synthesizer elements such as oscillators, filters, and amplifiers, or on sound production algorithms with a well defined principle of operation, neural audio synthesis implements deep neural networks (DNNs) that process large-scale set of audio data.

The method may be regarded as a mixture of two synthesis approaches: an analysis-resynthesis and a cross-synthesis. The output it produces is based on analysis of sampled sounds, which in majority of cases are musical instruments, with addition of some animal or otherwise environment-related sounds. Two such samples are combined into an interesting, and often unforeseen, mix of features of both.

Neural audio synthesis is based on two foundations: the NSynth large-scale dataset, and the WaveNet-like [582] autoencoder. DNNs train better when supplied with a large amount of high-quality data, therefore the NSynth dataset has been created. It consists of 306043 recordings of musical notes performed using 1006 separate instruments, each note with a defined pitch, timbre, and envelope, acquired from commercial sample libraries [182]. Samples last four seconds, where the note is held on for three, and the remaining one second is released. Recordings are monophonic, with $f_s = 16$ kHz. Pitches range over MIDI notes 21–108, and are sampled at five different velocities (25, 50, 75, 100, 127), however not all instruments were able to produce all combinations. Samples are attributed with annotations that are either based on human evaluation, or are the result of heuristic algorithms. The first annotation type is the source, either acoustic, electronic, or synthetic. The second assigns a sound to a particular family, such as bass, brass, organ, string, etc. The last one is a set of tags that define certain qualities, such as bright, dark, distortion, fast decay, long release, multiphonic, non-linear envelope, percussive, reverb, or tempo-synced.

The WaveNet is a generative approach to probabilistic modelling of a raw acoustic signal, and is based on the design of van den Oord et al. [582], where the next signal sample is predicted from a fixed-size input of prior sample values by a a stack of dilated convolutions. It attempts to capture longer term structure by learning temporal hidden codes. Solution presented by Engel et al. does not require external conditioning. A temporal encoder uses an input waveform to produce an *embedding* (a mathematical representation) $Z = f(x)$. The input is causally shifted and fed into the WaveNet decoder. The decoder reproduces the input with a joint probability of signal x [582]

$$p(x) = \prod_{i=1}^{N} p\left(x_i | x_1, \ldots, x_{N-1}, f(x)\right) \tag{3.38}$$

Given two recordings, the autoencoder extracts 16 temporal features of each, and interpolates them linearly to create new embeddings. The embeddings may be regarded as semantically meaningful hidden representation of sound features. They serve as high-level control data that manipulates tone, timbre, and dynamics. When decoded, they produce a sound that combines qualities of both input sounds.

Generative model implemented in the neural audio synthesis is able to fuse various aspects of the input instruments, and often yields perceptually interesting reconstruc-

tions. The output can capture expressive aspects related to timbre and dynamics of either of component signals, yet the result may be fairly distinct from any of input recordings. It has a tendency to exaggerate behaviour related to amplitude modulation. While the NSynth consists of separate notes, the method is somewhat capable of reproducing note transitions when supplied with sequence of pitch, dynamics and timbre data. It has to be considered though, that the technique, with regards to musical sound synthesis, is relatively new, and requires further research. In a manner that is becoming increasingly popular, and could accelerate method development, many of its elements, including the NSynth dataset, are well documented, open and easily accessible, allowing to perform independent tests or embed them in other projects.

3.2. Physical Modelling Methods

The majority of synthesis methods are focused on the sound itself. In their core they deal with sound either as a signal with its properties and parameters, or as a purposefully evoked auditory sensation with certain attributes. In either way they aim at designing a sound. Unlike them, physical modelling methods aim at designing a sound source, and specifically a source that simulates working principle, behaviour, and last but not least, way of controlling a musical instrument. They allow to use a virtual counterpart of existing or possible to conceive instrument that operates as a mathematical model based on physical description of the real object. The model predicts the output sound, while its inputs are designed to represent instrument controls and parameters [526].

Actually, one of the primary advantages that make physical modelling synthesis particularly appealing is a fact that its parameters are closely related to instrument control gestures [61], and thus meaningful for a performer. It may be exploited by pairing a synthesizer with a controller that mimics the actual instrument, e.g. violin-like, or trumpet-like, thereby allowing a performer to employ playing skills and experience acquired with a real instrument to produce an expressive performance. The remaining parameters of a model are intuitive as well, and correspond to physical features of an instrument, such as its geometry and material properties.

Contrary to sampling, which is able to produce realistic results only in response to separate, discrete excitation that simply triggers playback of a sound sample, physical modelling methods deal equally well with continuous control, such as bowing or blowing [526]. This makes them particularly suitable for real-time live performance, with direct human control, but may cause difficulties in automatic control, e.g. in sequencers. In such cases simulation of an instrument needs to be accompanied by simulation of a performer to fill-in a continuous control stream between a discrete sequence data.

Due to character of its control parameters and because a model simulates an internal state of an instrument, physical modelling synthesis has a unique property of producing broad selection of additional sound effects, including note-to-note transitions and various phenomena related to specific performance techniques, such as multi-

phonics in wind instruments. If a model is designed in elaborate detail, they need not be considered and simulated as separate cases, but will emerge if certain combination of input parameters is met. One of such phenomena is the re-excitation [526] which causes small, though audible alterations to sound produced when a vibrating string is struck again. More generally, physical modelling introduces interaction between a performer and an instrument.

Access to internal state of an instrument model leads to more control dimensions. For instance, in case of a string it is possible not only to adjust excitation force, but its spatial distribution and location as well. A simulated string may be plucked, struck or bowed at any point which changes output sound accordingly. Similarly, any point or combination of points within a model may be considered its output.

The idea of applying physical modelling techniques to synthesize sound of musical instruments is not new. Initial studies had been carried out by Ruiz [484], and more followed [235, 236, 29]. However, numerical modelling can be computationally expensive if a model involved is supposed to demonstrate realistic qualities. Therefore, unless some serious simplifications had been employed, physical modelling synthesis was difficult to implement in real-time applications, thereby missing its greatest advantage. In consequence, initial development was aimed at finding efficient methods, but later progress in computer technology, such as adaptation of multi-core central processing units as well as general-purpose computing on graphics processing units (GPGPU) [530, 254, 440, 63] provided enough processing power to carry out experiments with more general methods to simulate much subtler physical and auditory effects.

Physical modelling methods employ simplified instrument models that omit many properties of real objects they simulate. Therefore when the aim of a synthesis process is to obtain very close resemblance to a particular existing instrument, it may be hard to precisely replicate its timbre through modelling only. In such cases model data can be supplemented with measurements of the original object. Smith refers to such technique as **structured sampling** [526]. Like in signal sampling, a selected quantity is measured over some dimension, but it does not have to be acoustic pressure wave in time. Instead, an impulse response of an instrument body or similar data can be obtained to use with a model and bring its output closer to the original.

Although initially physical modelling synthesis was aimed at simulation of acoustic instruments only, over time it has been employed to model circuits of electroacoustic instruments and analogue synthesizers as well. The latter technique is referred to as virtual analog [573, 570, 149]. One may also attempt to create models of instruments that do not exist. An obvious reason would be to test if some hypothetical designs demonstrate required properties before building a real instrument. This, however, involves models much more complete and detailed than commonly used in sound synthesis [3]. Models of such complexity require far too much processing time to serve as a viable base for synthesis, even non real-time. On the other hand, playable models of non-existing instruments may directly serve as a means to produce new sounds, either by some new design decisions, or by allowing real-time control mechanisms impossible in real instruments, e.g. related to geometry or material. Still, even such models retain the fundamental advantage of physical modelling synthesis: parameters con-

trolled by a performer are related to some understandable physical features, therefore are intuitive and meaningful. According to Bilbao [61] physical modelling methods applied to produce sounds not corresponding to recognisable physical objects may be referred to as **quasi-physical synthesis**. The concept has been presented in works of Djoharian [166] or Leonard and Cadoz [333], and earlier suggested by Roads [470].

Modelling of a musical instrument for the synthesis purposes might be carried out using a number of methods. The most important difference between them concerns selection of physical phenomena they simulate and simplifications they employ. As a result, methods commonly referred to as physical modelling synthesis may markedly differ in complexity and requirements for processing capabilities. Some of them may require a laborious model design process. So far the most successful approaches include lumped models, modal synthesis, waveguide synthesis, various hybrid methods, and direct numerical simulations [470, 61, 526]. They make an extensive use of finite difference method (FDM) for solving differential equations, digital waveguides and digital filters for efficient modelling of various wave propagating structures [145], as well as modal techniques with Fourier decomposition as a bridge between additive synthesis and physical modelling.

3.2.1. Finite Difference Approximation

A musical instrument may be modelled with a set of partial differential equations. In a digital domain such equations can be approximated with finite differences. In the **finite difference method** (FDM) a spatial domain is restricted to a grid consisting of a finite number of points [61]. At these points values of a numerical solution are calculated and recursively advanced through discrete time steps. FDM is straightforward and may be applied to a variety of systems, linear as well as non-linear. In each time step a state of a system is known at any point of a grid, which allows to freely choose and change observation points. An immediate accessibility to any part of a model is crucial for the purpose of real-time synthesis, where a performer interacts with a model through some form of physical controller. Following a controller state, a state of a model is changed accordingly, and instantly. The main concern of the method is to ensure its numerical stability, which has been extensively researched and discussed by Bilbao [61].

Sound synthesis with FDM starts with a choice of modelled object, such as a string, a pipe, a bar, a membrane, a plate, etc. Depending on the object, an appropriate governing equation has to be chosen, and such choice is made on the basis of musical or structural acoustics. Many cases start with the wave equation and supplement it with additional terms that model some desired properties of the target object, like dumping or coupling with other objects. Due to digital nature of a model, a differential equation has to be approximated with a difference equation, which is reshaped into a **finite difference scheme**. A scheme allows to compute value of a solution in a given grid point, using its current, and some number of its previous values, as well as values obtained from points within its neighbourhood. Boundary conditions affect a scheme at some number of points on grid edges, while initial conditions determine values for the whole grid in a number of initial time steps. The model is 'played' by

applying excitation either in a form of another coupled distributed model, or – more often – a lumped element. In some applications excitation may even be simulated by a sampled data array.

To a large extent, the entire FDM section of this book is based on the book of Bilbao [61], in which he presented and thoroughly analysed a comprehensive set of difference operators used to design finite difference schemes for sound synthesis, along with numerous examples. Among other works related to the subject, the basic theory of the method has been discussed by Strikwerda [538], while a brief, simplified approach has been presented by Adib [5]. Another concise compendium on FDM in acoustics with some directions on scheme design can be found in works of Kristiansen and Viggen [309], and various paths of arriving at solution have been described by Causon and Mingham [106].

3.2.1.1. Temporal Operators

A continuous function of time $u(t)$ is approximated in discrete points $u(nT)$ with a *time series* $u[n]$, where integer n is the time index. T is the sampling period, or sampling interval

$$T = \frac{1}{f_s} \tag{3.39}$$

where f_s is the sampling frequency.

The fundamental operation on a time series is the shift. Shift operators are applied to the whole time series. They are defined as [61]

$$e_{t+}u[n] = u[n+1] \tag{3.40}$$

$$e_{t-}u[n] = u[n-1] \tag{3.41}$$

The former is a forward shift operator, and the latter – the backward shift. The identity operator is defined as [61]

$$1u[n] = u[n] \tag{3.42}$$

and it may be multiplied by a scalar.

While designing certain types of FD schemes, accuracy-affecting effects of various operators may be compensated with averaging operators that approximate identity operation [61]

$$\mu_{t+} \triangleq \frac{1}{2}(e_{t+} - 1) \cong 1 \tag{3.43}$$

$$\mu_{t-} \triangleq \frac{1}{2}(1 + e_{t-}) \cong 1 \tag{3.44}$$

$$\mu_{t\cdot} \triangleq \frac{1}{2}(e_{t+} + e_{t-}) \cong 1 \tag{3.45}$$

The first derivative operators may be approximated as follows [61]

$$\delta_{t+} \triangleq \frac{1}{T}(e_{t+} - 1) \cong \frac{\mathrm{d}}{\mathrm{d}t} \tag{3.46}$$

$$\delta_{t-} \triangleq \frac{1}{T}(1 - e_{t-}) \cong \frac{\mathrm{d}}{\mathrm{d}t} \tag{3.47}$$

$$\delta_{t\cdot} \triangleq \frac{1}{2T}(e_{t+} - e_{t-}) \cong \frac{\mathrm{d}}{\mathrm{d}t} \tag{3.48}$$

Due to temporal relations of time series it operates on, (3.46) is referred to as the forward difference approximation, (3.47) as the backward, and (3.48) as the centred difference approximation.

Basic operators defined in (3.43)–(3.48) can be combined to form other approximations, such as the second derivative

$$\delta_{tt} \triangleq \delta_{t+}\delta_{t-} = \frac{1}{T^2}(e_{t+} - 2 + e_{t-}) \cong \frac{\mathrm{d}^2}{\mathrm{d}t^2} \tag{3.49}$$

or a composition of averaging operators, which itself is also an averaging operator

$$\mu_{tt} \triangleq \mu_{t+}\mu_{t-} \tag{3.50}$$

as well as any other combination, e.g. $\mu_{t+}\delta_{t-}$, or $\mu_{t+}\mu_{t\cdot}\delta_{t\cdot}$. In some cases it can be advantageous to use a linear combination of operators, such as [61]

$$\alpha\delta_{t+} + (1 - \alpha)\delta_{t-} \tag{3.51}$$

where α is the parameter that can assume any real value.

Forward and backward difference operators defined in (3.43), (3.44), (3.46), and (3.47) have a temporal width of two, requiring two adjacent time steps for approximation (Fig. 3.41). They are first-order accurate [61]. Centred operators (3.45) and (3.48) as well as the second derivative approximation operator (3.49) span three time steps, thus they have a temporal width of three, and are second-order accurate. In most sound synthesis applications higher than second-order accuracy does not lead to perceptible gains, but strongly impacts performance. Therefore use of operators with temporal width larger than three is rare [61].

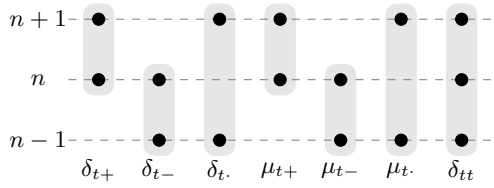


Figure 3.41. Temporal width of basic operators, assuming they operate at time step n
Source: author's elaboration, based on Bilbao [61]

3.2.1.2. Spatial Operators

Grid

Real musical instruments often assume complex geometric forms, yet not all their elements are equally relevant with regards to sound production mechanism. For the

synthesis purposes it is common to simulate only these elements, that are directly involved in sound production. Many of them can be modelled as one-dimensional (1D) or two-dimensional (2D) structures represented on grids.

A 1D grid function $u_l[n]$ approximates continuous function $u(x,t)$ at position $x = lX$ and time $t = nT$, where X is the spatial sampling interval, or grid spacing (Fig. 3.42).

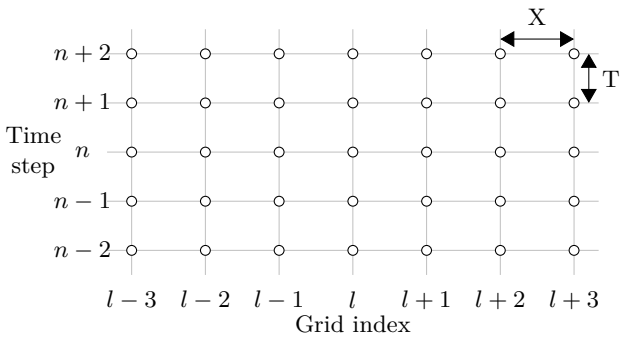


Figure 3.42. A grid for simulation of 1D elements; X is the spatial sampling interval, and T is the temporal sampling interval

2D elements can be conveniently simulated in Cartesian (x,y) or polar (r,θ) coordinates (Fig. 3.43). Both systems are related by the following expressions

$$\begin{aligned} r &= \sqrt{x^2+y^2} \\ \theta &= \arctan\frac{y}{x} \end{aligned} \tag{3.52}$$

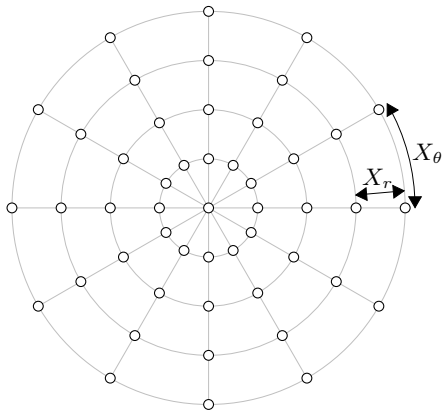


Figure 3.43. A single time frame of a 2D spatial grid in polar coordinates with spatial intervals X_r and X_θ

A grid function $u_{l,m}[n]$ in Cartesian coordinates approximates continuous function $u(x, y, t)$ at position $x = lX_x$, $y = mX_y$, and time $t = nT$, where X_x and X_y are spatial sampling intervals in x and y dimension, accordingly. In polar coordinates a grid function $u_{l,m}[n]$ represents continuous function $u(r, \theta, t)$ at position $r = lX_r$, $\theta = mX_\theta$, and time $t = nT$. While in some cases intervals X_x and X_y may be equal[7], intervals X_r and X_θ are generally different (Fig. 3.43).

1D Spatial Operators

Temporal shift operators applied to a grid do not differ from e_{t+} and e_{t-} defined in (3.40) and (3.41). They span over the entire grid, i.e. all values of spatial and temporal indices l and n, though they modify temporal index only. The same applies to operators approximating time derivative and identity, based on e_{t+} and e_{t-}, minding that on a spatial-temporal grid δ_{t+}, δ_{t-}, $\delta_{t\cdot}$, and δ_{tt} approximate partial time derivatives.

A set of new operators is required to handle spatial dimension. Forward and backward spatial shift operators are defined as [61]

$$e_{x+}u_l[n] = u_{l+1}[n] \tag{3.53}$$

$$e_{x-}u_l[n] = u_{l-1}[n] \tag{3.54}$$

Spatial averaging operators are analogous to temporal operators [61]

$$\mu_{x+} \triangleq \frac{1}{2}(e_{x+} - 1) \tag{3.55}$$

$$\mu_{x-} \triangleq \frac{1}{2}(1 + e_{x-}) \tag{3.56}$$

$$\mu_{x\cdot} \triangleq \frac{1}{2}(e_{x+} + e_{x-}) \tag{3.57}$$

and spatial derivative operators assume similar forms as well [61]

$$\delta_{x+} \triangleq \frac{1}{X}(e_{x+} - 1) \cong \frac{\partial}{\partial x} \tag{3.58}$$

$$\delta_{x-} \triangleq \frac{1}{X}(1 - e_{x-}) \cong \frac{\partial}{\partial x} \tag{3.59}$$

$$\delta_{x\cdot} \triangleq \frac{1}{2X}(e_{x+} - e_{x-}) \cong \frac{\partial}{\partial x} \tag{3.60}$$

$$\delta_{xx} \triangleq \delta_{x+}\delta_{x-} = \frac{1}{X^2}(e_{x+} - 2 + e_{x-}) \cong \frac{\partial^2}{\partial t^2} \tag{3.61}$$

Although it is uncommon to use higher than the second time derivative in sound synthesis, in more realistic models higher spatial derivatives may be encountered [61]

$$\delta_{xxxx} \triangleq \delta_{xx}\delta_{xx} = \frac{1}{X^4}(e_{x+}e_{x+} - 4e_{x+} + 6 - 4e_{x-} + e_{x-}e_{x-}) \cong \frac{\partial^4}{\partial t^4} \tag{3.62}$$

[7]Such simplification, where $X_x = X_y = X$ is reasonable and common in isotropic problems.

Some models require approximations of mixed derivatives, such as $\frac{\partial^3}{\partial t \partial x^2}$. Appropriate operators are constructed by combining simpler operators. In this case two variants are of particular interest [61], i.e.

$$\begin{aligned}\delta_{t-}\delta_{xx}u_l[n] = \frac{1}{TX^2}(u_{l+1}[n] - 2u_l[n] + u_{l-1}[n] - \\ - u_{l+1}[n-1] + 2u_l[n-1] - u_{l-1}[n-1])\end{aligned} \tag{3.63}$$

$$\begin{aligned}\delta_{t\cdot}\delta_{xx}u_l[n] = \frac{1}{2TX^2}(u_{l+1}[n+1] - 2u_l[n+1] + u_{l-1}[n+1] - \\ - u_{l+1}[n-1] + 2u_l[n-1] - u_{l-1}[n-1])\end{aligned} \tag{3.64}$$

Expressions (3.63) and (3.64) present mixed operators applied to a grid function $u_l[n]$. The important difference is the use of forward, i.e. unknown future values $n+1$ in (3.64) (Fig. 3.44). Such operators lead to implicit schemes that involve linear system solution techniques, but often result in better behaving, more stable models.

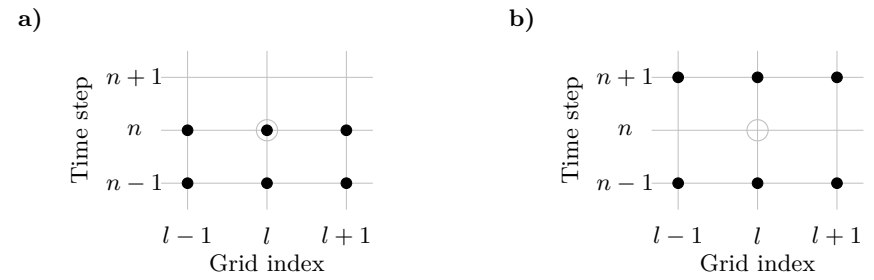


Figure 3.44. Stencils of two variants of operators approximating mixed spatial-temporal derivatives: $\delta_{t-}\delta_{xx}$ (a) and $\delta_{t\cdot}\delta_{xx}$ (b); black dots symbolise grid points used for approximation, and a gray circle indicates the approximated point

2D Spatial Operators

Two-dimensional spatial shift operators in Cartesian coordinates assume the following form [61]

$$\begin{aligned} e_{x+}u_{l,m}[n] &= u_{l+1,m}[n] & e_{y+}u_{l,m}[n] &= u_{l,m+1}[n] \\ e_{x-}u_{l,m}[n] &= u_{l-1,m}[n] & e_{y-}u_{l,m}[n] &= u_{l,m-1}[n] \end{aligned} \tag{3.65}$$

Shift operators are used to define forward, backward and centred difference operators approximating the first spatial derivative in Cartesian coordinates [61]

$$\begin{aligned} \delta_{x+} &\triangleq \frac{1}{X_x}(e_{x+} - 1) \cong \frac{\partial}{\partial x} & \delta_{y+} &\triangleq \frac{1}{X_y}(e_{y+} - 1) \cong \frac{\partial}{\partial y} \\ \delta_{x-} &\triangleq \frac{1}{X_x}(1 - e_{x-}) \cong \frac{\partial}{\partial x} & \delta_{y-} &\triangleq \frac{1}{X_y}(1 - e_{y-}) \cong \frac{\partial}{\partial y} \\ \delta_{x\cdot} &\triangleq \frac{1}{2X_x}(e_{x+} - e_{x-}) \cong \frac{\partial}{\partial x} & \delta_{y\cdot} &\triangleq \frac{1}{2X_y}(e_{y+} - e_{y-}) \cong \frac{\partial}{\partial y} \end{aligned} \tag{3.66}$$

Two-dimensional averaging operators, as well as operators approximating second derivatives $\frac{\partial^2}{\partial x^2}$, $\frac{\partial^2}{\partial y^2}$, $\frac{\partial^2}{\partial x \partial y}$, and other combined operators are defined in a similar manner.

In polar coordinates shift operators are given by [61]

$$\begin{aligned} e_{r+}u_{l,m}[n] &= u_{l+1,m}[n] & \qquad e_{\theta+}u_{l,m}[n] &= u_{l,m+1}[n] \\ e_{r-}u_{l,m}[n] &= u_{l-1,m}[n] & \qquad e_{\theta-}u_{l,m}[n] &= u_{l,m-1}[n] \end{aligned} \tag{3.67}$$

which leads to the following forms of difference operators approximating spatial derivatives [61]

$$\begin{aligned} \delta_{r+} &\triangleq \frac{1}{X_r}(e_{r+} - 1) \cong \frac{\partial}{\partial r} & \qquad \delta_{\theta+} &\triangleq \frac{1}{X_\theta}(e_{\theta+} - 1) \cong \frac{\partial}{\partial \theta} \\ \delta_{r-} &\triangleq \frac{1}{X_r}(1 - e_{r-}) \cong \frac{\partial}{\partial r} & \qquad \delta_{\theta-} &\triangleq \frac{1}{X_\theta}(1 - e_{\theta-}) \cong \frac{\partial}{\partial \theta} \\ \delta_{r\cdot} &\triangleq \frac{1}{2X_r}(e_{r+} - e_{r-}) \cong \frac{\partial}{\partial r} & \qquad \delta_{\theta\cdot} &\triangleq \frac{1}{2X_\theta}(e_{\theta+} - e_{\theta-}) \cong \frac{\partial}{\partial \theta} \end{aligned} \tag{3.68}$$

Apart from operators approximated by (3.66) and (3.68) two derivative operators working in two spatial dimensions are crucial for physical modelling synthesis. The first is the Laplacian, in Cartesian coordinates defined as

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \tag{3.69}$$

and in polar coordinates

$$\Delta u = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial u}{\partial r}\right) + \frac{1}{r^2} + \frac{\partial^2 u}{\partial \theta^2} \tag{3.70}$$

The second is the bi-Laplacian, also referred to as the biharmonic operator [61]

$$\Delta\Delta u = \frac{\partial^4 u}{\partial x^4} + 2\frac{\partial^4 u}{\partial x^2 \partial y^2} + \frac{\partial^4 u}{\partial y^4} \tag{3.71}$$

For the purpose of sound synthesis Laplacian in Cartesian coordinates is usually approximated using a five-point operator that either uses grid points adjacent to the centre [61]

$$\delta_{\Delta\boxplus} = \delta_{xx} + \delta_{yy} \tag{3.72}$$

or uses grid points adjacent diagonally [61]

$$\delta_{\Delta\boxtimes} = \delta_{xx} + \delta_{yy} + \frac{X^2}{2}\delta_{xx}\delta_{yy} \tag{3.73}$$

Both can be combined into a nine-point operator with a free parameter α [61]

$$\delta_{\Delta\alpha} = \alpha\delta_{\Delta\boxplus} + (1 - \alpha)\delta_{\Delta\boxtimes} \tag{3.74}$$

which might be adjusted to control isotropy. Similarly, various approximations of bi-Laplacian are possible, from a basic one

$$\delta_{\Delta\boxplus\Delta\boxplus} \triangleq \delta_{\Delta\boxplus}\delta_{\Delta\boxplus} \tag{3.75}$$

to more elaborate parametrised combinations. Stencils of Laplacian and bi-Laplacian approximations are presented in Figure 3.45.

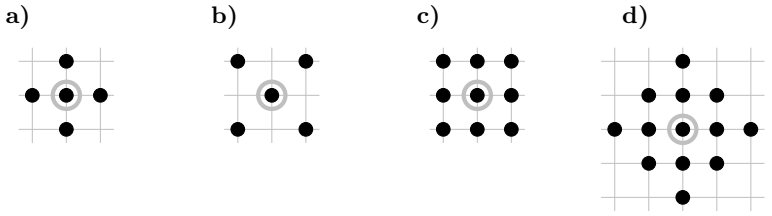


Figure 3.45. Stencils of three variants of operators approximating Laplacian, $\delta_{\Delta\boxplus}$ (a), $\delta_{\Delta\boxtimes}$ (b), and $\delta_{\Delta\alpha}$ (c), and an approximation of bi-Laplacian $\delta_{\Delta\boxplus\Delta\boxplus}$ (d); black dots symbolise grid points used for approximation, and a gray circle indicates the approximated point

Discrete approximation of the Laplacian in polar coordinates may be given by [61]

$$\delta_{\Delta\circ}u = \frac{1}{r}\delta_{r+}\left(\left(\mu_{r-}r\right)\delta_{r-}u\right) + \frac{1}{r^2}\delta_{\theta\theta}u \tag{3.76}$$

where $\delta_{\theta\theta} = \delta_{\theta+}\delta_{\theta-}$. A different form is required for a centre point, where $l = 0$ [61]

$$\delta_{\Delta\circ}u_{0,0} = \frac{4}{N_\theta X_r^2}\sum_{m=0}^{N_\theta-1}\left(u_{1,m} - u_{0,0}\right) \tag{3.77}$$

where N_θ is the total number of grid points with the same radius[8].

Bi-Laplacian in polar coordinates can be approximated using the following difference operator [61]

$$\delta_{\Delta\circ,\Delta\circ}u_{l,m} = \delta_{\Delta\circ}\delta_{\Delta\circ}u_{l,m} \qquad 2 \le l \le N_r - 2 \tag{3.78}$$

Again, a different form is required at the grid centre ($l = 0$) and in points around it ($l = 1$) [61]

$$\delta_{\Delta\circ,\Delta\circ}u_{0,0} = \frac{16}{3N_\theta X_r^4}\sum_{m=0}^{N_\theta-1}\left(u_{2,m} - 4u_{1,m} + 3u_{0,0}\right) \tag{3.79}$$

$$\delta_{\Delta\circ,\Delta\circ}u_{1,m} = \frac{4}{N_\theta X_r^2}\sum_{m=0}^{N_\theta-1}\delta_{\Delta\circ}u_{1,m} - \delta_{\Delta\circ}u_{0,0} \tag{3.80}$$

[8]The domain is periodic, i.e. looped in the θ dimension, thus grid points with indices m and (m mod N_θ) are the same.

Points Beyond Grid

Difference operators make use of values in adjacent points to calculate approximation at some grid location. Examples of stencils are shown in Figures 3.44 and 3.45. However, a grid is finite and on or near its ends, depending on stencil, some required points do not exist. Such values have to be obtained using boundary conditions. The conditions depend on modelled element, but some very common and simple cases are conditions of Dirichlet type and Neumann type.

A good example is the operator δ_{xx} approximating second derivative in 1D (3.49), used in many finite difference schemes based on the wave equation. When applied to a grid function u_l it uses locations $(l-1)$, l, and $(l+1)$

$$\delta_{xx} u_l = \frac{1}{X^2}\left(u_{l+1} - 2u_l + u_{l-1}\right) \tag{3.81}$$

If the domain size is N, then the grid contains elements $[0, 1, \ldots, N-1]$. Approximations for grid points in the range of $[1, \ldots, N-2]$ are easily obtained, but in $l = 0$ and in $l = (N-1)$ locations from outside grid, $l = -1$ and $l = N$ respectively, are required. Bilbao refers to such points as *virtual* or *image* grid points (Fig. 3.46).

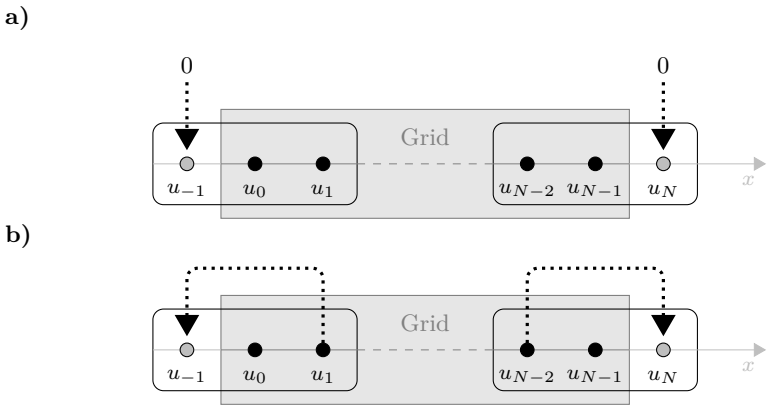


Figure 3.46. Points beyond grid, or virtual grid points, required by a second spatial derivative difference operator δ_{xx}; gray box and black dots represent a 1D spatial grid and grid points; operator stencil for points u_0 and u_{N-1} on both grid ends is marked with black frames; gray dots represent virtual points; in (a) values outside grid are set to 0, according to Dirichlet conditions; in (b) values are copied from appropriate points inside, according to Neumann conditions with first derivative approximated by centred difference operator δ_x.

Dirichlet-type boundary conditions set values of grid function outside a grid to zero or constant. They can represent either a fixed string end or an open end of a pipe. In this particular case it is simply $u_{-1} = 0$ and $u_N = 0$, and therefore

$$\begin{aligned} \delta_{xx} u_0 &= \frac{1}{X^2}(u_1 - 2u_0) \\ \delta_{xx} u_{N-1} &= \frac{1}{X^2}(-2u_{N-1} + u_{N-2}) \end{aligned} \tag{3.82}$$

Neumann-type boundary conditions set first spatial derivative to zero outside a grid: $\frac{\partial}{\partial x}u_{-1} = 0$ and $\frac{\partial}{\partial x}u_N = 0$. It may be associated to a closed end of a pipe, or to rather abstract case of a string end able to move only in a transverse direction. If the first derivative is approximated with the centred difference operator (3.60) it yields $u_{-1} = u_1$ and $u_N = u_{N-2}$, leading to

$$\begin{aligned} \delta_{xx}u_0 &= \frac{2}{X^2}(u_1 - u_0) \\ \delta_{xx}u_{N-1} &= \frac{2}{X^2}(-u_{N-1} + u_{N-2}) \end{aligned} \tag{3.83}$$

Other first derivative approximations are also possible, although they may result in different expressions for virtual grid points.

In a 2D grid function $u_{l,m}$ with N_x and N_y grid points in x and y dimension respectively, Dirichlet conditions are applied similarly to the 1D case. Function values for points outside a grid are set to zero for $l < 0$, $l > (N_x - 1)$, $m < 0$, and $m > (N_y - 1)$. A more interesting case involves Neumann conditions and 5-point Laplacian operator $\delta_{\Delta\boxplus}$ (3.74). On the edges the condition is

$$\begin{aligned} \delta_{x-}u_{0,m} &= 0 \qquad & \delta_{x+}u_{N_x-1,m} &= 0 \\ \delta_{y-}u_{l,0} &= 0 \qquad & \delta_{y+}u_{l,N_y-1} &= 0 \end{aligned} \tag{3.84}$$

This modifies the expression for the Laplacian on the edge ($l = 0$), considering $u_{-1,m} = u_{0,m}$

$$\delta_{\Delta\boxplus}u_{0,m} = \frac{1}{X^2}(u_{0,m+1} + u_{0,m-1} + u_{1,m} - 3u_{0,m}) \tag{3.85}$$

Expressions for the remaining edges are similarly obtained, though corners need to be considered separately, for instance in $l = 0$ and $m = 0$

$$\delta_{\Delta\boxplus}u_{0,0} = \frac{1}{X^2}(u_{0,1} + u_{1,0} - 2u_{0,0}) \tag{3.86}$$

Wider operator stencils introduce a larger number of virtual grid points. Such cases require more boundary conditions [61].

3.2.1.3. Input and Output Operators

Output Operators

In the most basic situation one chooses a single point from within a FD grid used for physical modelling synthesis, and collects its values in subsequent time steps to produce an output waveform. A point value in a single step becomes one signal sample. Any number of such points may be observed to form separate output channels. However, an output may have to be taken from a position between grid points, or – and it is a particularly interesting case – the output is moving. In the latter case many intermediate positions between grid points may need to be observed in order to avoid discontinuous changes in the output produced. Such values can be obtained through interpolation.

In some cases it may be enough to simply truncate position of the observation point. For 1D problem and grid function u_l at a fixed point in time, and observation point x_o, a leftward truncation is carried out as follows

$$l_o = \left\lfloor \frac{x_o}{X} \right\rfloor \tag{3.87}$$

where $\lfloor \cdot \rfloor$ is the *floor* function. Therefore the simplest interpolation operator is given by [61]

$$I_0(x_o)u_l = u_{l_o} \tag{3.88}$$

An improved operator applies linear interpolation and is defined as [61]

$$I_1(x_o)u_l = (1-\alpha_o)u_{l_o} + \alpha_o u_{l_o+1} \tag{3.89}$$

where α_o is the remainder of truncation

$$\alpha_o = \frac{x_o}{X} - l_o \tag{3.90}$$

Further improvements are possible if necessary, such as Lagrange cubic interpolation [61]

$$\begin{aligned} I_3(x_o)u_l = &\frac{\alpha_o(\alpha_o-1)(\alpha_o-2)}{-6}u_{l_o-1} + \frac{(\alpha_o-1)(\alpha_o+1)(\alpha_o-2)}{2}u_{l_o} + \\ &+ \frac{\alpha_o(\alpha_o+1)(\alpha_o-2)}{-2}u_{l_o+1} + \frac{\alpha_o(\alpha_o+1)(\alpha_o-1)}{6}u_{l_o+2} \end{aligned} \tag{3.91}$$

Operator I_0 requires only one grid point to work, while I_1 uses two, and I_3 as much as four, as shown in Figure 3.47.

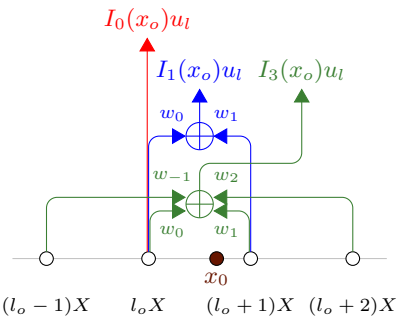


Figure 3.47. Three interpolation operators over a one-dimensional grid: truncation I_0 (red), linear I_1 (blue), and cubic I_3 (green); coefficients w_i represent grid point weights, according to (3.89) and (3.91)
Source: author's elaboration, based on Bilbao [61]

In a 2D problem a grid function $u_{l,m}$ is truncated in both dimensions

$$l_o = \left\lfloor \frac{x_o}{X_x} \right\rfloor \qquad m_o = \left\lfloor \frac{y_o}{X_y} \right\rfloor \tag{3.92}$$

The simplest interpolation operator is defined as [61]

$$I_0(x_o, y_o)u_{l,m} = u_{l_o,m_o} \tag{3.93}$$

Bilinear interpolation operator assumes the following form [61]

$$\begin{aligned} I_1(x_o, y_o)u_{l,m} =&(1-\alpha_{x,o})(1-\alpha_{y,o})u_{l_o,m_o} + (1-\alpha_{x,o})\alpha_{y,o}u_{l_o,m_o+1} + \\ &+ \alpha_{x,o}(1-\alpha_{y,o})u_{l_o+1,m_o} + \alpha_{x,o}\alpha_{y,o}u_{l_o+1,m_o+1} \end{aligned} \tag{3.94}$$

where $\alpha_{x,o}$ and $\alpha_{y,o}$ are remainders of truncation

$$\alpha_{x,o} = \frac{x_o}{X_x} - l_o \qquad \alpha_{y,o} = \frac{y_o}{X_y} - m_o \tag{3.95}$$

Some systems may require interpolation of entire grid. For two 1D grid functions, u_l and v_m, with different grid point spacings, $x_l = lX_u$ and $x_m = mX_v$, and assuming $X_v \leq X_u$, the upsampling operation is performed using interpolant I_p of order p according to the following expression

$$v_m = I_p(x_m)u_l \tag{3.96}$$

which may be symbolised as [61]

$$v = \mathcal{I}_{X_u \to X_v, p}u \tag{3.97}$$

For the purpose of downsampling one may use a conjugate interpolant [61]

$$\mathcal{I}^*_{X_v \to X_u, p} = \frac{X_u}{X_v}\mathcal{I}^T_{X_u \to X_v, p} \tag{3.98}$$

Symbol $\mathcal{I}_{X_u \to X_v, p}$ represents a rectangular matrix with size and contents depending on boundary conditions.

Input Operators

Just like an instrument modelled needs to produce output, it has to receive an input as well. Two most obvious situations are excitation and interconnection. In the first one a distributed element, such as a string, has to be excited to produce a sound. In the second one two distributed elements are interconnected, like a string and a soundboard. In both situations an input may be described in terms of its location and spatial distribution.

The most basic distribution for 1D problems can be modelled with a Dirac delta function $\delta(x - x_i)$, where the input is located in a single point x_i obtained through truncation

$$l_i = \left\lfloor \frac{x_i}{X} \right\rfloor \tag{3.99}$$

The distribution of an input is therefore expressed as the following grid function [61]

$$J_{l,0}(x_i) = \begin{cases} \dfrac{1}{X} & l = l_i \\ 0 & l \neq l_i \end{cases} \tag{3.100}$$

and may also be referred to as spreading operator [61].

Higher order 1D spreading operators use the fractional part α_i, analogous to the remainder α_o in interpolation operators (3.90)

$$\alpha_i = \frac{x_i}{X} - l_i \tag{3.101}$$

Linear spreading distribution has the following form [61]

$$J_{l,1}(x_i) = \frac{1}{X} \begin{cases} 0 & l < l_i \\ (1-\alpha_i) & l = l_i \\ \alpha_i & l = l_i + 1 \\ 0 & l > l_i + 1 \end{cases} \tag{3.102}$$

A smoother, cubic spreading distribution is defined as [61]

$$J_{l,3}(x_i) = \frac{1}{X} \begin{cases} 0 & l < l_i - 1 \\ -\frac{1}{6}\alpha_i(\alpha_i - 1)(\alpha_i - 2) & l = l_i - 1 \\ \frac{1}{2}(\alpha_i - 1)(\alpha_i + 1)(\alpha_i - 2) & l = l_i \\ -\frac{1}{2}\alpha_i(\alpha_i + 1)(\alpha_i - 2) & l = l_i + 1 \\ \frac{1}{6}\alpha_i(\alpha_i + 1)(\alpha_i - 1) & l = l_i + 2 \\ 0 & l > l_i + 2 \end{cases} \tag{3.103}$$

For the purpose of spreading operators in 2D problems truncation and fractional parts are expressed as

$$l_i = \left\lfloor \frac{x_i}{X_x} \right\rfloor \qquad m_i = \left\lfloor \frac{y_i}{X_y} \right\rfloor \tag{3.104}$$

$$\alpha_{x,i} = \frac{x_i}{X_x} - l_i \qquad \alpha_{y,i} = \frac{y_i}{X_y} - m_i \tag{3.105}$$

The first and second order 2D spreading grid functions assume the following forms [61]

$$J_{l,m,0}(x_i, y_i) = \begin{cases} \dfrac{1}{X_x X_y} & l = l_i,\ m = m_i \\ 0 & \text{elsewhere} \end{cases} \tag{3.106}$$

$$J_{l,m,1}(x_i, y_i) = \frac{1}{X_x X_y} \begin{cases} (1-\alpha_{x,i})(1-\alpha_{y,i}) & l = l_i,\ m = m_i \\ (1-\alpha_{x,i})\alpha_{y,i} & l = l_i,\ m = m_i + 1 \\ \alpha_{x,i}(1-\alpha_{y,i}) & l = l_i + 1,\ m = m_i \\ \alpha_{x,i}\alpha_{y,i} & l = l_i + 1,\ m = m_i + 1 \\ 0 & \text{elsewhere} \end{cases} \tag{3.107}$$

3.2.1.4. Simplified Ideal String

The most basic model of a string is based on 1D wave equation

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} \tag{3.108}$$

In case of a string, $u(x,t)$ represents the string displacement, and c is the transverse wave velocity, given as

$$c = \sqrt{\frac{T_0}{\rho A}} \tag{3.109}$$

where T_0 represents the string tension, ρ is its density, and A it the area of its cross-section. Such model does not include effects associated with loss or stiffness, nor coupling with excitation element, such as hammer.

For the purpose of simulation it is convenient to substitute x with a scaled, dimensionless coordinate $\tilde{x}$ [61]

$$\tilde{x} = \frac{x}{L} \tag{3.110}$$

where L is the length of a string, and c with scaled γ

$$\gamma = \frac{c}{L} \tag{3.111}$$

With $\tilde{x}$ and γ the wave equation assumes the following form

$$\frac{\partial^2 u}{\partial t^2} = \gamma^2 \frac{\partial^2 u}{\partial \tilde{x}^2} \tag{3.112}$$

In further expressions containing γ symbol, tilde will be omitted, and x will be assumed scaled.

The wave equation (3.112) may be approximated with the following finite difference scheme [61]

$$\delta_{tt} u_l[n] = \gamma^2 \delta_{xx} u_l[n] \tag{3.113}$$

where δ_{tt} and δ_{xx} are difference operators, as in (3.49) and (3.81). The scheme can be expanded as follows

$$\frac{1}{T^2}\left(u_l[n+1] - 2u_l[n] + u_l[n-1]\right) = \frac{\gamma^2}{X^2}\left(u_{l+1}[n] - 2u_l[n] + u_{l-1}[n]\right) \tag{3.114}$$

By introducing a parameter referred to as the **Courant number** [61]

$$\lambda \triangleq \frac{\gamma T}{X} \tag{3.115}$$

(3.114) changes into a recursion

$$u_l[n+1] = 2(1-\lambda^2)u_l[n] + \lambda^2(u_{l-1}[n] + u_{l+1}[n]) - u_l[n-1] \tag{3.116}$$

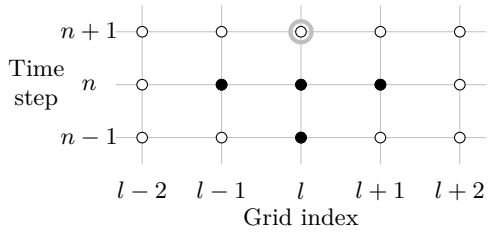


Figure 3.48. A stencil of finite difference scheme given by (3.116); computed grid point is indicated by a gray circle, points used by the scheme are represented by black dots; new value in step $(n+1)$ depends on two previous time steps: three locations, left $(l-1)$, middle l, and right $(l+1)$ are used in step n, and only middle one in step $(n-1)$

Stencil of the scheme (3.116) is shown in Figure 3.48.

The most basic **boundary condition** for a string is of a Dirichlet type, applied on both ends of a grid. It represents a string fixed on both ends. Values of grid function in virtual grid points are therefore set to zero. No additional condition is required due to scheme (3.116) being of second order in space. Though it is not the only choice for a string. Bilbao mentions also lossy boundary conditions. Such conditions for the left string end assume the form [61]

$$\frac{\partial}{\partial t}u(0,t) = \alpha\frac{\partial}{\partial x}u(0,t) \tag{3.117}$$

where $\alpha > 0$ is some chosen constant.

The scheme is second order in time, therefore it requires two initial conditions, such as [61]

$$\begin{aligned} u(x,0) &= u_0(x) \\ \frac{\partial}{\partial t}u(x,0) &= v_0(x) \end{aligned} \tag{3.118}$$

Initial conditions may be utilised as an excitation of a string model. For an idealised strike condition initial displacement $u_0(x)$ can be set to zero, and initial velocity $v_0(x)$ to some spatial distribution. An idealised pluck condition may be modelled conversely, with initial velocity set to zero, and initial displacement set to a spatial distribution. As Bilbao points out, one of common choices for a distribution is the triangular function [61]

$$c_{tri}(x) = \begin{cases} \dfrac{c_0}{x_0}x & 0 \le x \le x_0 \\ \dfrac{c_0}{x_0 - 1}(x-1) & x_0 < x \le 1 \end{cases} \tag{3.119}$$

Another such choice is the raised cosine function [61]

$$c_{rc}(x) = \begin{cases} \frac{c_0}{2}\left(1 + \cos\left(\frac{\pi(x - x_0)}{x_{hw}}\right)\right) & |x - x_0| \leq x_{hw} \\ 0 & |x - x_0| > x_{hw} \end{cases} \tag{3.120}$$

where c_0 is the peak displacement located at x_0, and x_{hw} is the half-width of a pulse.

For such a simple string model a choice of initial conditions is the primary means of timbre control. Detailed relationship between characteristics of an initial condition distribution and resultant signal spectrum is widely discussed in musical acoustics [191, 47, 92]. Both distributions are shown in Figure 3.49 with spectra produced when a distribution is applied as 'plucked' condition, i.e. it generates values of $u_0(x)$. Figure 3.50 presents a time evolution for both distributions, each applied as plucked and struck initial condition for the wave equation (3.108). Apart from altering initial conditions parameters, additional timbre adjustments may be carried out by moving readout position along a string (Fig. 3.51).

Modelling an excitation through initial conditions is not close enough to physical behaviour for more advanced models. In such models excitation is implemented in a scheme through a term representing an applied force.

A basic mechanism to control fundamental frequency f_0, and hence pitch of a signal produced by scheme (3.116) is to adjust the value of γ. From (3.111), and since for an ideal string f_0 is produced by the lowest mode of length $2L$, one obtains

$$\gamma = 2f_0 \tag{3.121}$$

A finite difference scheme produces only an approximation to a solution of real wave equation. Its behaviour may be analysed using various methods. An extensive discussion of two such methods and their application to numerous finite difference schemes is given by Bilbao [61]. In case of scheme (3.116) stability is obtained when

$$\lambda \leq 1 \tag{3.122}$$

which is the Courant–Friedrichs–Lewy (CFL) condition [140]. Since the scheme is explicit, i.e. it calculates future state from current and previous states, the right side of the inequality has a value of 1.

Approximation errors introduced by finite difference schemes usually cause phase and group velocity to differ from values given by a continuous model. The extent of such difference is referred to as **numerical dispersion** and in distributed systems it may have a consequence of warping component frequencies in produced signal [61]. In case of scheme (3.116) numerical dispersion does not occur if $\lambda = 1$. Therefore, it is advantageous to get as close as possible to this value. Since λ relates time step T with grid spacing X for a given γ (3.115), and in digital sound processing time step is usually fixed by a given sampling frequency, once a new γ value is set, a new grid spacing X should be established. As a consequence, grid size may change (Fig. 3.52).

Expression (3.113) is a simple, but not the only possible finite difference scheme approximating 1D wave equation. Though it is often the best solution due to its property of being exact for $\lambda = 1$, other, more elaborate schemes, discussed by Dablain,

Noye, Tuomela, and Bilbao [408, 147, 409, 565, 59], can be of use for designing difference schemes representing more complex systems.

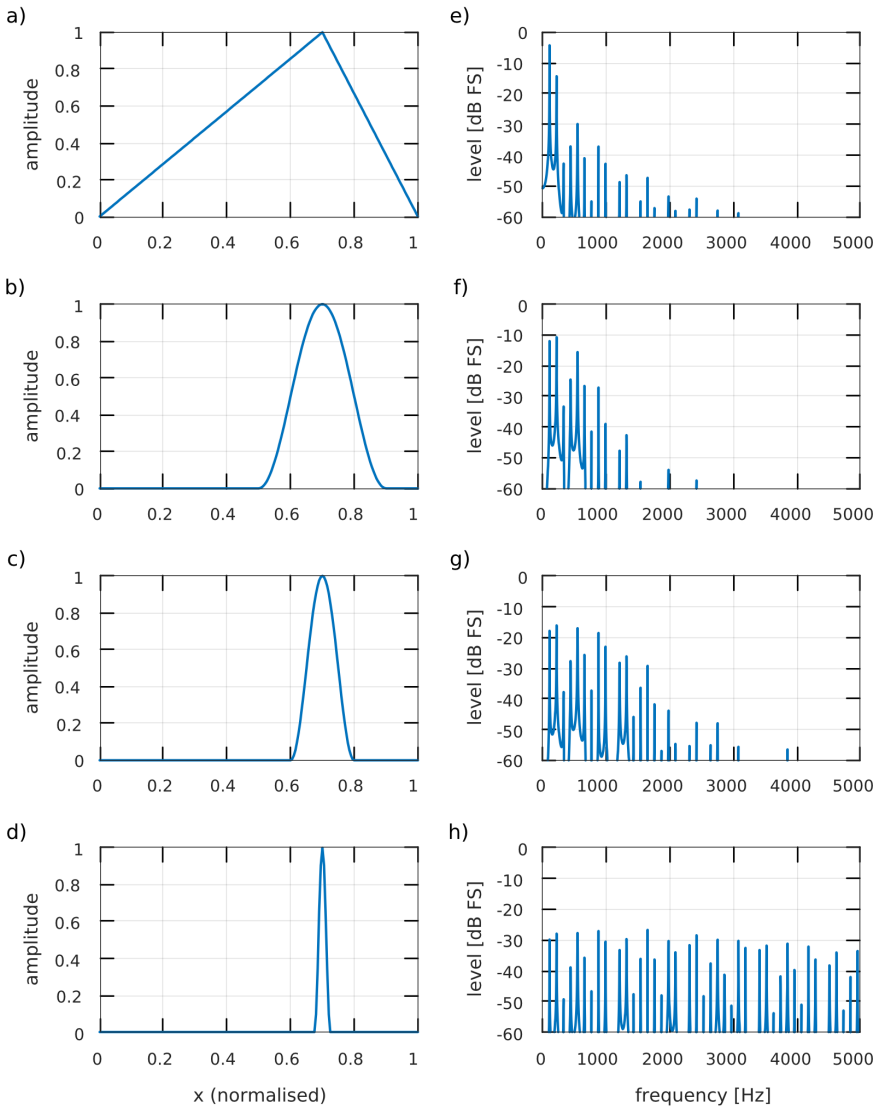


Figure 3.49. Examples of spatial distributions used to set initial conditions for ideal plucking or striking of a string: triangular (a), and raised cosine with various half-width values, $x_{hw} = 0.4$ (b), $x_{hw} = 0.2$ (c), and $x_{hw} = 0.05$ (d); plots (e–f) present signal spectra obtained when a respective distribution is applied to pluck a string tuned for 110 Hz

Two specific schemes introduce a parameter that allows to control their properties. In the first one identity is extended using controlled amount of centred averaging operator $(\alpha + (1 - \alpha)\mu_{x\cdot})$ [61]

$$\delta_{tt} u_l[n] = \gamma^2 (\alpha + (1 - \alpha)\mu_{x\cdot})\delta_{xx} u_l[n] \tag{3.123}$$

where α is the free parameter.

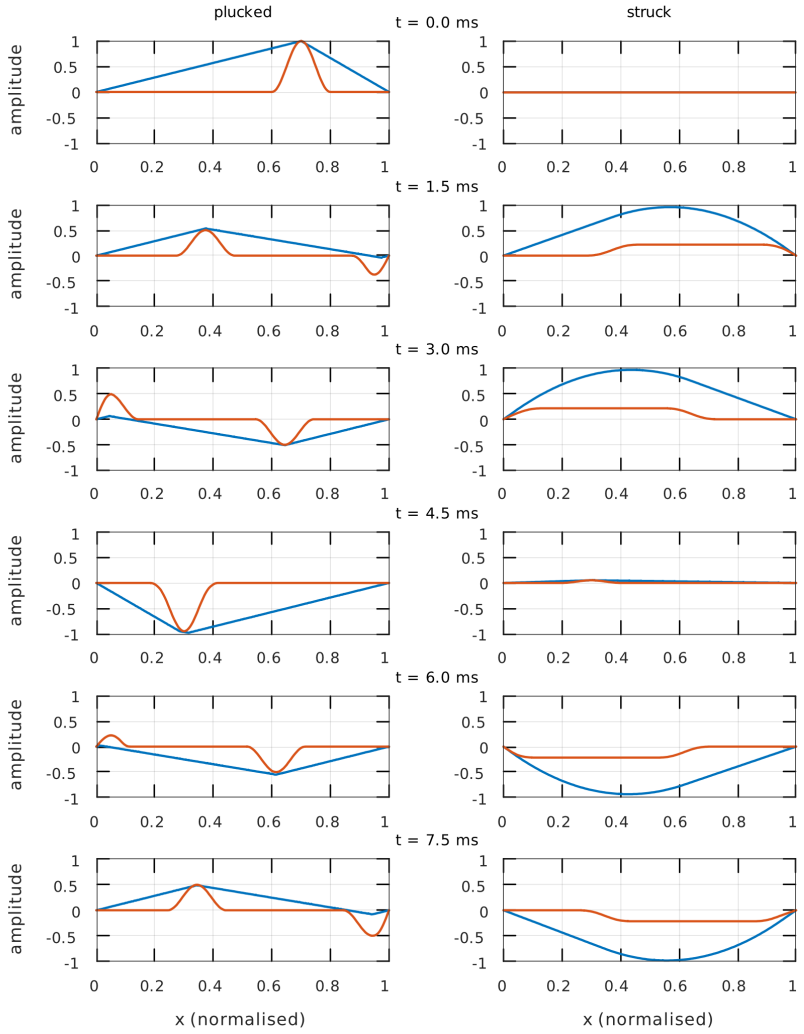


Figure 3.50. Time evolution of the wave equation solution for two types of initial conditions, each with two spatial distributions: triangle (blue), and raised cosine (red); in both cases $f_0 = 110\ Hz$ and $x_0 = 0.7$; for raised cosine $x_{hw} = 0.2$; for a plucked condition initial peak displacement $u_0 = 1$ and initial peak velocity $v_0 = 0$; for a struck condition initial peak displacement $u_0 = 0$ and initial peak velocity $v_0 = 950$

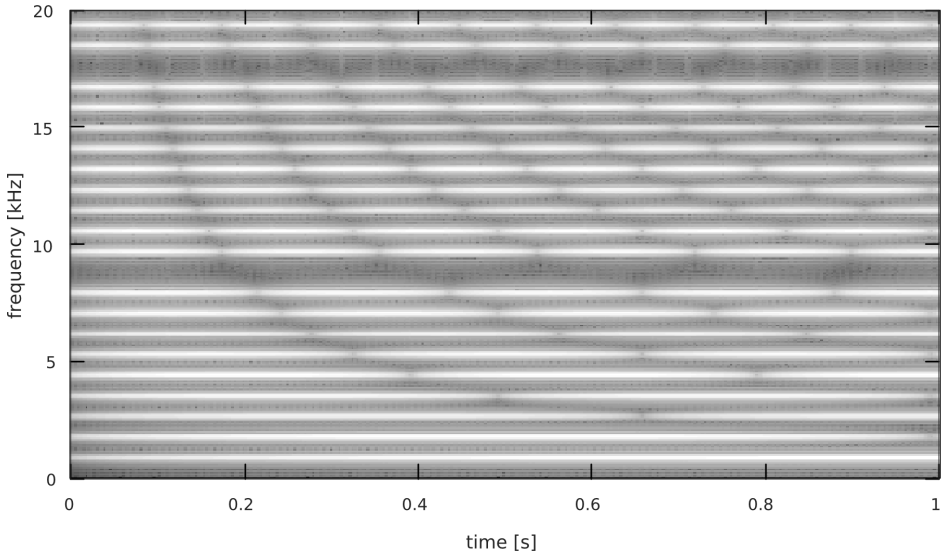


Figure 3.51. Spectral evolution of a signal produced by 880 Hz string scheme (3.116) for raised cosine plucked initial condition with $x_{hw} = 0.1$ and $x_0 = 0.7$; the evolution is caused by linear shift of readout position from 0 to $\frac{L}{2}$ using linear interpolation (3.89)

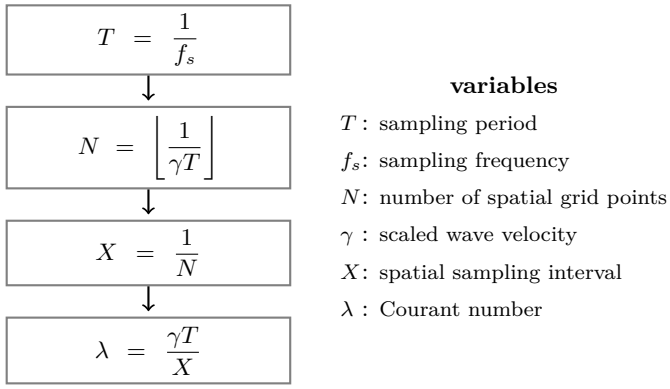


Figure 3.52. Method of setting grid parameters in scheme (3.116) aimed at satisfying CFL condition as close to 1 as possible; the assumption is that sampling period is fixed by chosen sampling frequency, hence grid tuning is applied to spatial sampling interval and number of spatial grid points
Source: author's elaboration, based on Bilbao [61]

Stencil of such scheme is wider then the one from (3.113) – it uses two points on each side of approximated grid point (Fig. 3.53a), therefore it requires additional numerical boundary condition.

Analysis of the scheme brings condition for the α parameter, as well as condition for λ, which now is related to α [61]

$$\lambda \leq \begin{cases} \sqrt{8(1-\alpha)} & \frac{1}{2} \leq \alpha \leq \frac{3}{4} \\ \frac{1}{\sqrt{2\alpha - 1}} & \alpha \geq \frac{3}{4} \end{cases} \tag{3.124}$$

If $\alpha = 1$ scheme (3.123) transforms into (3.113).

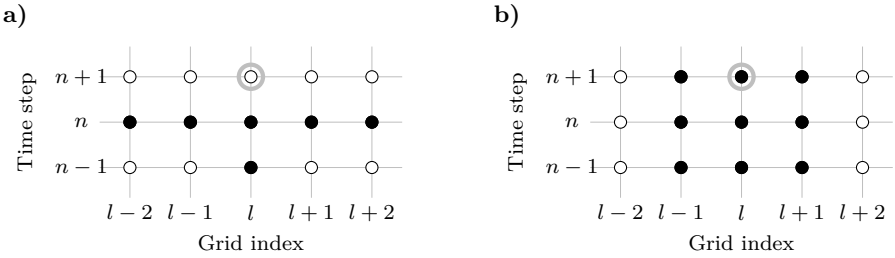


Figure 3.53. A stencil of finite difference scheme (3.123) (a) and (3.123) (b); black dots symbolise grid points used for approximation; a gray circle indicates the approximated point

A different approach is represented by an implicit scheme [61]

$$(\theta + (1-\theta)\mu_{x\cdot})\delta_{tt} u_l[n] = \gamma^2 \delta_{xx} u_l[n] \tag{3.125}$$

where θ is a free parameter. If $\theta = 1$ scheme (3.125) transforms into (3.113). For other values scheme is implicit – it couples three adjacent grid point values in future step $(n+1)$ (Fig. 3.53b) – and requires linear system solution techniques [61]. There are, however, fast techniques, such as the Thomas algorithm [549] that do not require full matrix inversion and can be applied in this particular case. The free parameter is bounded by condition $\theta \geq \frac{1}{2}$, and λ is related to θ [61]

$$\lambda \leq \sqrt{2\theta - 1} \tag{3.126}$$

Implicit schemes, such as the one given by (3.125) allow λ to become larger then 1 for larger θ. Therefore, when operating with the same sampling frequency, implicit schemes allow for lower numerical dispersion than explicit schemes and are generally more stable [61].

A very simple model, like an ideal string, allows some degree of control:

- pitch through wave velocity,
- dynamics through excitation parameters,
- and timbre, mostly through excitation parameters, but also by choosing readout position.

Even though a number of controllable quantities may seem small, it can be mapped to a larger number of physical properties. E.g. wave velocity is controlled through

the scaled γ parameter which includes string length L (3.111), and the velocity itself is calculated from a given string tension, density, and thickness (3.109). Such model, however, is not the most interesting synthesis target. It lacks properties of real strings that make their sound and its control distinct and appealing. Two of them are damping and stiffness, and they can be introduced by extending (3.108) by additional terms.

3.2.1.5. Damped Stiff String

The simplest model of a string with **dissipation** is given by the following expression [61]

$$\frac{\partial^2 u}{\partial t^2} = \gamma^2 \frac{\partial^2 u}{\partial x^2} - 2\sigma_0 \frac{\partial u}{\partial t} \tag{3.127}$$

where σ_0 is a non-negative constant. An appropriate finite difference scheme may be obtained by extending (3.113) [61]

$$\delta_{tt} u_l[n] = \gamma^2 \delta_{xx} u_l[n] - 2\sigma_0 \delta_{t\cdot} u_l[n] \tag{3.128}$$

which results in the following recursion

$$\begin{aligned} u_l[n+1] = \frac{2}{1+\sigma_0 T}\left((1-\lambda^2)u_l[n] + \frac{\lambda^2}{2}(u_{l-1}[n] + u_{l+1}[n])\right) - \\ - \frac{1-\sigma_0 T}{1+\sigma_0 T} u_l[n-1] \end{aligned} \tag{3.129}$$

As of physical interpretation, constant σ_0 may be responsible for various phenomena, such as radiation or internal losses. The effect is frequency independent, though it causes dispersive wave propagation (Fig. 3.54). Considering σ_0 as one of control parameters it is convenient to express it in relation to 60 dB decay time T_{60} [61]

$$T_{60} = \frac{6\ln(10)}{\sigma_0} \tag{3.130}$$

Strings in musical instruments demonstrate a small degree of **stiffness** which causes a slight inharmonicity in produced spectra. The effect is particularly audible in piano [47]. A basic model of string with stiffness may be given by adding fourth order term to the wave equation [568]

$$\frac{\partial^2 u}{\partial t^2} = \gamma^2 \frac{\partial^2 u}{\partial x^2} - \kappa^2 \frac{\partial^4 u}{\partial x^4} \tag{3.131}$$

where κ is the stiffness parameter. It is defined as [61]

$$\kappa = \sqrt{\frac{EI}{\rho A L^4}} \tag{3.132}$$

where E is the Young's modulus, I is the moment of inertia, ρ is the material density, A is the area of cross-section, and L is the length.

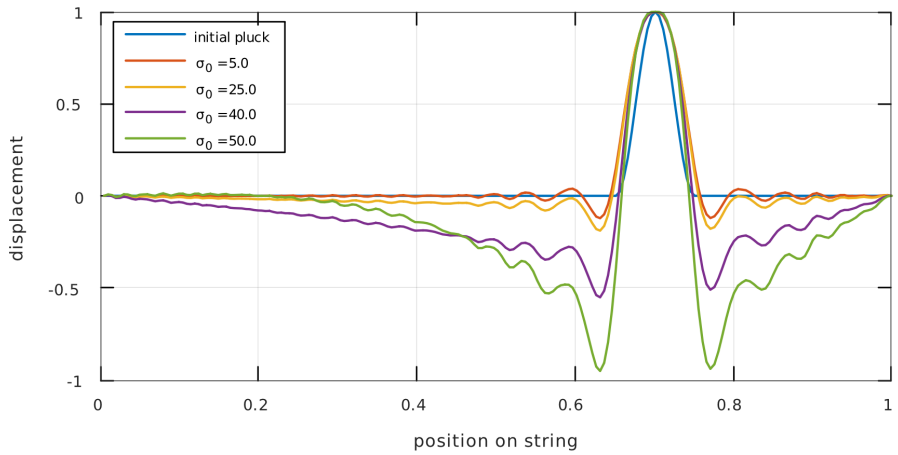


Figure 3.54. State of a 110 Hz string modelled by (3.129) after 2 seconds of simulation for different values of σ_0 loss parameter; initial condition was an ideal pluck with raised cosine distribution, $x_{hw} = 0.1$ and $x_0 = 0.7$; in order to compare shape of the displacement, the amplitude is normalised to 1 throughout all values of σ_0

The most direct approach to design a finite difference scheme for a stiff string will result with the following explicit scheme [61]

$$\delta_{tt} u_l[n] = \gamma^2 \delta_{xx} u_l[n] - \kappa^2 \delta_{xxxx} u_l[n] \tag{3.133}$$

The stability condition that relates spatial and temporal grid spacing assumes the following form [61]

$$X \geq \sqrt{\frac{\gamma^2 T^2 + \sqrt{\gamma^4 T^4 + 16\kappa^2 T^2}}{2}} \tag{3.134}$$

which constitutes a minimum spatial grid spacing for a given sampling frequency. The effect of altering parameter κ is shown in Figure 3.55.

As an alternative, Bilbao gives an example of implicit θ-type scheme [61]

$$(\theta + (1-\theta)\mu_{x\cdot})\, \delta_{tt} u_l[n] = \gamma^2 \delta_{xx} u_l[n] - \kappa^2 \delta_{xxxx} u_l[n] \tag{3.135}$$

though he states that it is more difficult to establish a value of θ that would be satisfactory in the entire frequency spectrum than in case of (3.125).

More realistic string model should not only include losses and stiffness, as given by (3.127) and (3.131), but also some kind of frequency dependence for the losses. Although natural spectral evolution of vibrating string may be rather complex, it is a realistic assumption that loss increases with frequency. Such behaviour may be modelled with the following partial differential equation [50]

$$\frac{\partial^2 u}{\partial t^2} = \gamma^2 \frac{\partial^2 u}{\partial x^2} - \kappa^2 \frac{\partial^4 u}{\partial x^4} - 2\sigma_0 \frac{\partial u}{\partial t} + 2\sigma_1 \frac{\partial^3 u}{\partial t \partial x^2} \tag{3.136}$$

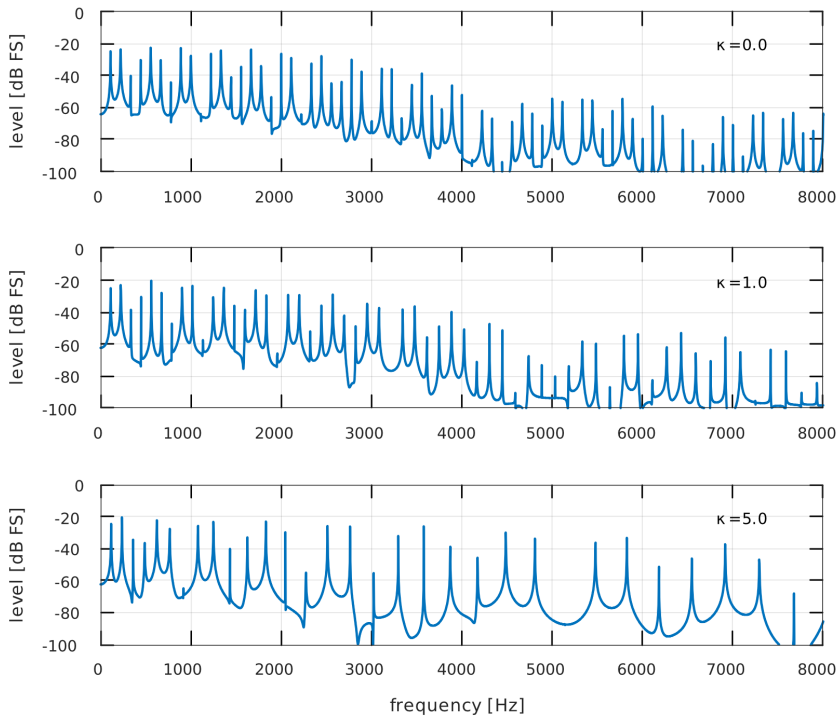


Figure 3.55. Spectrum of a signal produced by a 110 Hz string modelled using (3.133) with different values of stiffness parameter κ; stiffness causes detuning (raising) of partial frequencies; initial condition was an ideal pluck with raised cosine distribution, $x_{hw} = 0.1$ and $x_0 = 0.7$; each value of κ required recalculation of spatial grid size according to (3.134)

A practical guide for establishing both loss parameters, σ_0 and σ_1, is given by Bilbao, under the assumption of small losses. The parameters in question are set on the basis of two decay times T_{60} specified for two selected frequencies $f_1 < f_2$ [61]

$$\begin{aligned} \sigma_0 &= \frac{6\ln(10)}{\xi(f_2) - \xi(f_1)} \left(\frac{\xi(f_2)}{T_{60}(f_1)} - \frac{\xi(f_1)}{T_{60}(f_2)} \right) \\ \sigma_1 &= \frac{6\ln(10)}{\xi(f_2) - \xi(f_1)} \left(-\frac{1}{T_{60}(f_1)} + \frac{1}{T_{60}(f_2)} \right) \end{aligned} \tag{3.137}$$

where ξ is defined as

$$\xi(f) = \frac{-\gamma^2 + \sqrt{\gamma^4 + 16\kappa^2\pi^2 f^2}}{2\kappa^2} \tag{3.138}$$

One explicit differential scheme that approximates (3.136), assumes the following form, as given by Bilbao [61]

$$\delta_{tt} u_l[n] = \gamma^2 \delta_{xx} u_l[n] - \kappa^2 \delta_{xxxx} u_l[n] - 2\sigma_0 \delta_{t\cdot} u_l[n] + 2\sigma_1 \delta_{t-} \delta_{xx} u_l[n] \tag{3.139}$$

A change of operator δ_{t-} to $\delta_{t\cdot}$ in the mixed derivative term makes the scheme implicit [61]

$$\delta_{tt}u_l[n] = \gamma^2\delta_{xx}u_l[n] - \kappa^2\delta_{xxxx}u_l[n] - 2\sigma_0\delta_{t\cdot}u_l[n] + 2\sigma_1\delta_{t\cdot}\delta_{xx}u_l[n] \tag{3.140}$$

The result of simulation using the explicit scheme is presented in Figure 3.56.

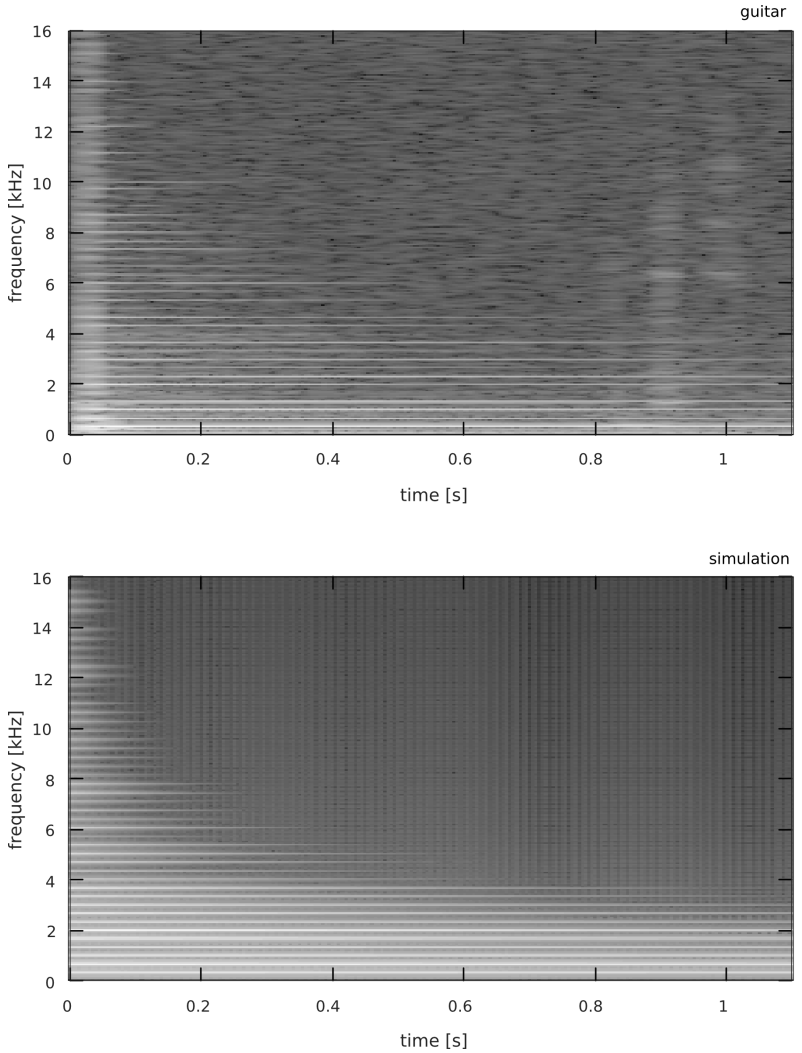


Figure 3.56. Recording of a real guitar string E4 compared to the result of simulation of a stiff string with frequency dependent loss obtained using explicit scheme (3.139) for raised cosine plucked initial condition with $x_{hw} = 0.3$ and $x_0 = 0.25$; loss parameters were set to $T_{60} = 8$ s at $f = 20$ Hz and $T_{60} = 0.15$ s at $f = 10$ kHz, and inharmonicity $\kappa = 2.05$

Stencil of difference schemes that include fourth order term makes it necessary to provide additional boundary conditions. For a clamped string it is

$$u = \frac{\partial u}{\partial x} = 0 \tag{3.141}$$

and in case of numerical approximation on the left end of a string:

$$u = \delta_{x+} u_0 = 0 \tag{3.142}$$

3.2.1.6. String Excitation

In majority of string instruments a string is excited by bowing, striking, or plucking. Excitation by **bowing** has been studied since Helmholtz [47, 191], and is still a subject of active research [359, 610, 352, 353, 200, 93].

For an ideal string excited by bowing in a single location x_i, the wave equation needs to be expanded into [61]

$$\frac{\partial^2 u}{\partial t^2} = \gamma^2 \frac{\partial^2 u}{\partial x^2} - \delta(x - x_i)\tilde{F}_B \phi(v_{rel}) \tag{3.143}$$

where:

$$v_{rel} = \frac{\partial u(x_i)}{\partial t} - v_B \tag{3.144}$$

$\tilde{F}_B = \tilde{F}_B(t) = \frac{F_B(t)}{m_s} \geq 0$ is the bow force divided by the string mass, $v_B = v_B(t)$ is the bow velocity, ϕ is the characteristic of friction, and $\delta(x - x_i)$ is the Dirac delta centred at x_i. Excitation location may also change over time, i.e. $x_i = x_i(t)$.

Characteristic ϕ has to satisfy the following conditions [61]

$$\phi(\eta)\eta \geq 0 \quad \text{or} \quad \operatorname{sgn}(\phi(\eta)) = \operatorname{sgn}(\eta) \tag{3.145}$$

which ensures its passivity, and it has to be bounded

$$|\phi| \leq 1 \tag{3.146}$$

Typically, friction characteristic is antisymmetric about $\eta = 0$ and includes two regimes, responsible for sticking and sliding. The former is represented by a steep positive slope in the centre. The latter, outside of the former, is represented by a softer negative slope. Bilbao gives three examples of such functions [61], as presented in Figure 3.57. The first one has a hard transition

$$\phi(\eta) = \operatorname{sgn}(\eta) e^{-\alpha|\eta|} \tag{3.147}$$

the second one is similar, but limiting value of sliding friction is non-zero

$$\phi(\eta) = \operatorname{sgn}(\eta)(\epsilon + (1 - \epsilon)e^{-\alpha|\eta|}) \tag{3.148}$$

and the last one has a soft transition

$$\phi(\eta) = \sqrt{2\alpha}\eta e^{-\alpha\eta^2 + \frac{1}{2}} \tag{3.149}$$

where the parameters $\alpha > 0$, and $0 < \epsilon < 1$.

Expression (3.143) may be approximated with the following finite difference scheme [61]

$$\begin{aligned} \delta_{tt} u_l[n] &= \gamma^2 \delta_{xx} u_l[n] - J(x_i[n]) \tilde{F}_B[n] \phi(r_{rel}) \\ v_{rel} &= I(x_i[n]) \delta_{t\cdot} u_l[n] - v_B[n] \end{aligned} \tag{3.150}$$

where $\tilde{F}_B[n] > 0$ and $v_B[n]$ are time series that may originate from a controller. Interpolation operator I is used to establish string velocity in a bowing point, and spreading operator J allows to distribute bow force over a string. In addition to CFL condition, another one is required to ensure uniqueness of the solution [61]

$$\lambda \leq \frac{2\gamma}{-\max(F_B)\min(\phi')} \tag{3.151}$$

Even though the scheme is implicit, it can be updated explicitly, once the relative velocity v_{rel} has been established.

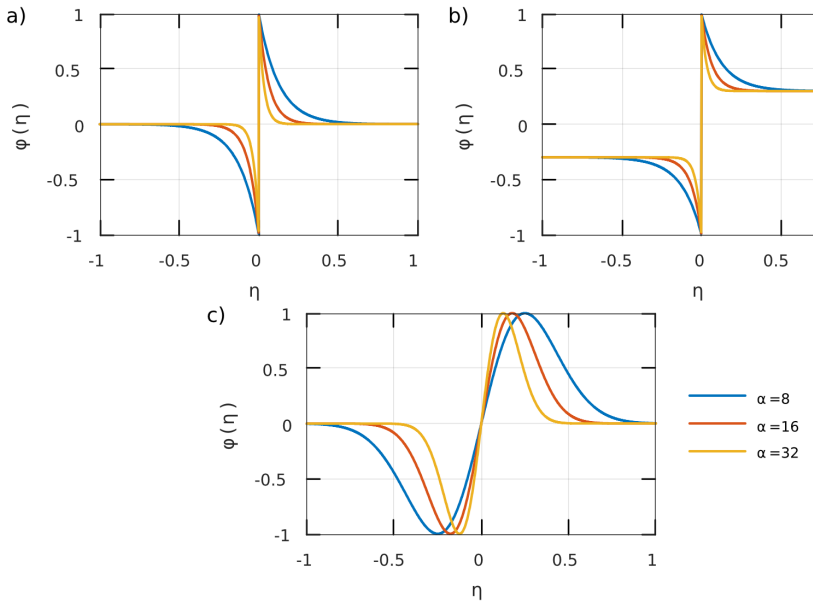


Figure 3.57. Three examples of friction characteristics, given by (3.147) (a), (3.148) (b), and (3.149) (c), for three values of α parameter, and $\epsilon = 0.3$
Source: author's elaboration, based on Bilbao [61]

The model of a bowed string may be further extended. Gillian [206] and Bavu [42] investigated torsional wave propagation. Pitteroff and Woodhouse studied a case of bow contact area [437]. Both extensions have been investigated in the context of sound synthesis by Serafin et al. [508, 507, 506].

Excitation by **striking** a string with a mallet or hammer can be modelled on the basis of a power law [108, 61]

$$F(u) = \omega_c^{\alpha+1} \mathrm{sgn}(u) \, |u|^{\alpha} \tag{3.152}$$

where the non-linear exponent α and stiffness parameter ω_c are established through measurements of real instruments or various laboratory test stands [420].

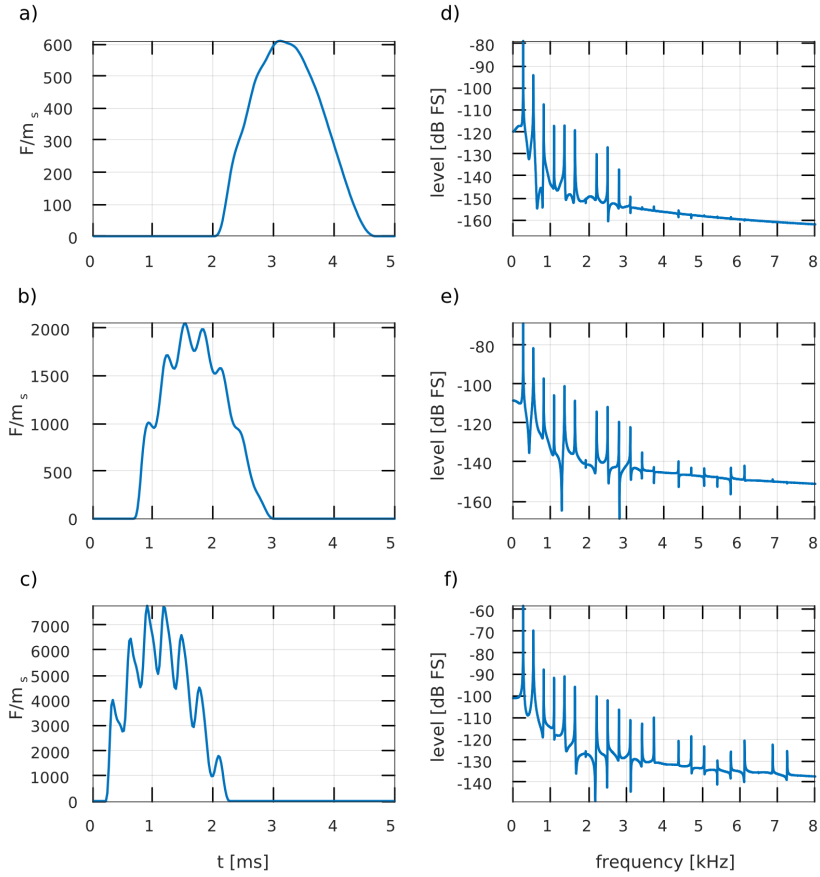


Figure 3.58. Force profiles (a–c) and respective spectra (d–f) for a hammer striking a string with three different initial velocities: 0.5 $\frac{m}{s}$ (a), 1.5 $\frac{m}{s}$ (b), and 5 $\frac{m}{s}$ (c); initial distance between colliding objects was set to 0.001 of string length; simulation was carried out using combination of two implicit schemes: for a stiff string with frequency dependent loss (3.139) and for a string coupled with hammer (3.162) and (3.163); hammer parameters were set, after Bilbao [61], to: $\mathcal{M} = 0.75$, $\omega_H = 0.75$, $\alpha = 2.5$, and its spatial profile was approximated by a raised cosine distribution with $x_{hw} = 0.05$ and $x_c = 0.12$; string parameters were set to: $f_0 = 262$ Hz, $\kappa = 5.27$, and loss to $T_{60} = 10$ s at 100 Hz and $T_{60} = 8$ s at 1000 Hz

If collision model is linear, contact time between a string and striking object does not depend on striking velocity. In such model striking velocity affects only

signal amplitude. If a model is non-linear, contact duration depends on velocity, which causes alteration of sound timbre (Fig. 3.58), characteristic for many string instruments.

The collision is one-sided, i.e. the force acts only if the displacement is positive, which may be indicated with the following notation

$$[F(u)]^{+} = \begin{cases} 0 & u \leq 0 \\ F(u) & u > 0 \end{cases} \tag{3.153}$$

A hammer striking an ideal string with power law characteristic may be expressed as [61]

$$\rho A \frac{\partial^2 u}{\partial t^2} = T_0 \frac{\partial^2 u}{\partial x^2} + \epsilon(x) F \tag{3.154}$$

where ρ is the string density, A is the area of string cross-section, T_0 is the string tension, and $\epsilon(x)$ is the spatial profile of a hammer. The force F assumes the following form [61]

$$F = -M_H \frac{\mathrm{d}^2 u_H}{\mathrm{d}t^2} = K_H \left([u_H - \langle \epsilon, u \rangle_{\mathcal{D}}]^{+} \right)^{\alpha} \tag{3.155}$$

where u_H is the hammer position over a non-displaced string, M_H is the hammer mass, and K_H is its stiffness parameter. Parameter α is usually set between 1.5 and 3.5 [539]. Operation $\langle \epsilon, u \rangle_{\mathcal{D}}$ indicates the *inner product* over a spatial domain $\mathcal{D}$, in this particular case a string, and is defined as [61]

$$\langle p[n], q[n] \rangle_{\mathcal{D}} = \sum_{l \in \mathcal{D}} X p_l[n] q_l[n] \tag{3.156}$$

where $p[n]$ and $q[n]$ are grid functions, and the result of the operation is a time series. The accompanying norm is defined as [61]

$$\|p[n]\|_{\mathcal{D}} = \sqrt{\langle p[n], p[n] \rangle_{\mathcal{D}}} \geq 0 \tag{3.157}$$

For simulation purposes it is more convenient to apply spatial scaling analogous to (3.110) and (3.111), which sets the domain to the unit interval $\mathcal{D} = \mathbb{U}$, and to introduce two parameters. The first one

$$\mathcal{M} = \frac{M_H}{\rho A L} \tag{3.158}$$

is the ratio of two colliding masses, hammer and string. The second one

$$\omega_H = \left(\frac{K_H}{M_H} \right)^{\frac{1}{1+\alpha}} \tag{3.159}$$

also characterises a hammer and has a meaning similar to frequency. Using these, (3.154) assumes the following form

$$\frac{\partial^2 u}{\partial t^2} = \gamma^2 \frac{\partial^2 u}{\partial x^2} + \epsilon(x)\mathcal{M}\tilde{F} \tag{3.160}$$

and the scaled force is expressed as

$$\tilde{F} = -\frac{\mathrm{d}^2 u_H}{\mathrm{d}t^2} = \omega_H^{\alpha+1} \left(\left[u_H - \langle \epsilon, u \rangle_{\mathbb{U}} \right]^+ \right)^{\alpha} \tag{3.161}$$

The system described by (3.160) and (3.161) may be approximated using the following explicit scheme [61]

$$\delta_{tt} u_l[n] = \gamma^2 \delta_{xx} u_l[n] + \epsilon \mathcal{M} \tilde{F} \tag{3.162}$$

$$\tilde{F} = -\delta_{tt} u_H[n] = \omega_H^{\alpha+1} \left(\left[u_H[n] - \langle \epsilon, u_l[n] \rangle_{\mathbb{U}_N} \right]^+ \right)^{\alpha} \tag{3.163}$$

where the domain $\mathbb{U}_N = [0, 1, \ldots, N]$. A simulation that presents the effect of a combination of explicit collision scheme and scheme for a stiff string with frequency dependent loss has been illustrated in Figure 3.58, with respective parameters.

A semi-implicit scheme, differing from the explicit one in the force part [61]

$$\begin{aligned} \tilde{F} &= -\delta_{tt} u_H[n] = \\ &= \omega_H^{\alpha+1} \left(\left[u_H[n] - \langle \epsilon, u_l[n] \rangle_{\mathbb{U}_N} \right]^+ \right)^{\alpha-1} \mu_{t\cdot} \left(u_H[n] - \langle \epsilon, u_l[n] \rangle_{\mathbb{U}_N} \right) \end{aligned} \tag{3.164}$$

can be utilised to avoid problems with numerical oscillation that may emerge while using a purely explicit scheme [61].

As in case of bow, more sophisticated models of collision between a mallet or a hammer and a string exist, and may be applied. For instance, it is possible to enhance the model with an effect of hysteresis [539]. Further discussion regarding modelling collisions of hammers and strings can be found in works of Hall [229, 230].

3.2.1.7. String Model Refinements

A string model with loss and stiffness is still a very basic one, and may be further refined. Firstly, it can be supplemented with additional elements, or integrated into more complex systems. Secondly, mechanisms adapted from musical and structural acoustics help to design new, or to improve existing models, so that they imitate the physical original in greater detail. Therefore, they are able to produce a larger set of characteristic effects, often utilised by performing musicians.

In the largest part of its playing range, except the lowest bass notes, piano strings are struck simultaneously in triplets or pairs tuned in unison. It is a source of some distinctive auditory effects, and can be quite simply simulated. The most important property of a string triplet is its non-ideal unison. The unison is slightly detuned, of the order of single cents [47], which causes beating and takes part in forming of a characteristic compound decay curve due to impedance-related effects[9] [191].

[9]The final decay rate is several times lesser then the initial decay rate.

The simplest model of a **string unison** involves a system of M partial differential equations, for instance, 1D wave equations

$$\frac{\partial^2 u_q}{\partial t^2} = (\gamma_q)^2 \frac{\partial^2 u_q}{\partial x^2} \qquad q = 1, \ldots, M \tag{3.165}$$

where M is the number of strings, and u_q is the transverse displacement of q-th string. All strings are assumed to be of equal length, and since γ is related to fundamental frequency, unison detuning is simulated by varying values of γ_q. Bilbao proposes the following expression [61]

$$\gamma_q = 2^{1+\frac{(2q-1-M)D}{2400(M-1)}} f_0 \qquad q = 1, \ldots, M \tag{3.166}$$

where D is the pitch difference in cents between the highest and the lowest-pitched string within the unison.

Expression (3.165) may be approximated with the following finite difference scheme [61]

$$\delta_{tt} u_{q,l}[n] = (\gamma_q)^2 \delta_{xx} u_{q,l}[n] \qquad q = 1, \ldots, M \tag{3.167}$$

Stability condition, such as CFL, has to be calculated for each string separately, though it will result in slightly different grid spacings. Alternatively, and it is the simpler solution, the greatest value of grid spacing X_{max} can be assumed for the whole unison. It will, however, cause all but one string to work away from Courant limit, reducing bandwidth of their signals.

The model of unison needs to be supplemented with hammer interaction [61]

$$\frac{\partial^2 u_q}{\partial t^2} = (\gamma_q)^2 \frac{\partial^2 u_q}{\partial x^2} - 2\sigma_0 \frac{\partial u_q}{\partial t} + \epsilon_q(x) \mathcal{M}_q \tilde{F}_q \qquad q = 1, \ldots, M \tag{3.168}$$

$$\frac{\mathrm{d}^2 u_H}{\mathrm{d}t^2} = -\sum_q \tilde{F}_q \qquad \tilde{F}_q = \omega_H^{\alpha+1} \left(\left[u_H[n] - \langle \epsilon_q, u_{q,l}[n] \rangle_{\mathbb{U}} \right]^+ \right)^\alpha \tag{3.169}$$

A simple loss with parameter σ_0 has been added, so that the effect of compound decay curve might be observed. Finally, the output of the unison model may be obtained through a simple sum at location x_o using interpolation operator of a chosen order [61]

$$u_o = \sum_{q=1}^{M} I(x_{o,q}) u_q \tag{3.170}$$

A comparison between a signal produced by one string and a three-string unison is shown in Figure 3.59.

Physical sound synthesis may be applied to simulate special performance techniques, such as **preparation of piano**. The preparation involves attaching – or putting close to strings – various objects, so that they come in contact when the string vibrates. The purpose of the procedure is to obtain new sound qualities from known instruments.

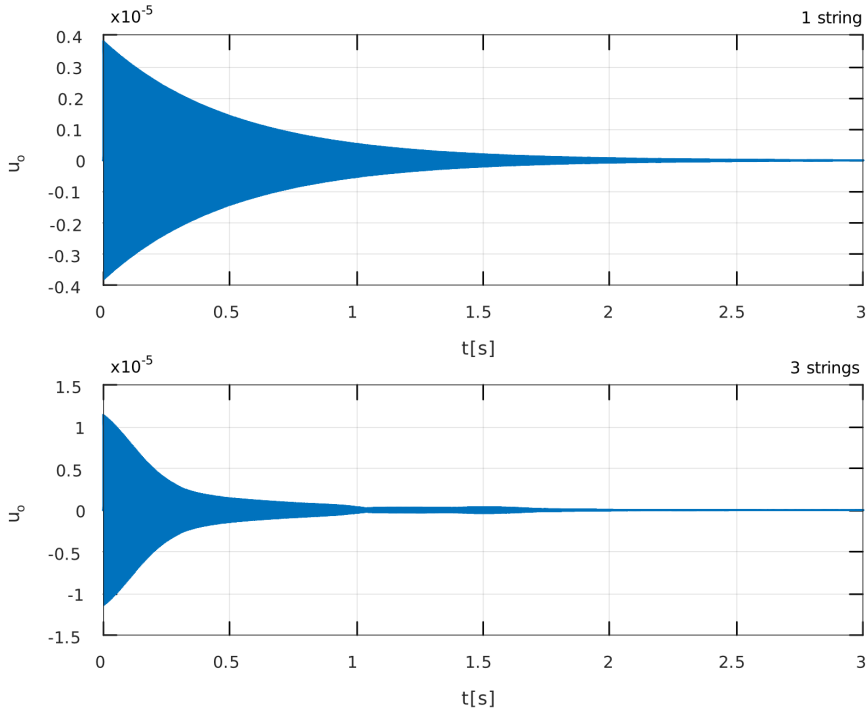


Figure 3.59. A comparison between output signal produced by one string and a three-string unison with characteristic compound decay curve; simulation was carried out using a model given by (3.168) and (3.169); simulation parameters were set to: average $f_0 = 220$ Hz, string detuning $D = 5$ ct, loss $T_{60} = 7$ s, hammer mass ratio $\mathcal{M} = 0.01$, hammer stiffness $\omega_H = 2500$, $\alpha = 2.5$; hammer spatial profile was approximated by a raised cosine distribution with $x_{hw} = 0.05$ and $x_c = 0.12$; output was obtained at $x_o = 0.3$

The basis for the model of prepared string may be the following expression [61]

$$\frac{\partial^2 u}{\partial t^2} = \gamma^2 \frac{\partial^2 u}{\partial x^2} - 2\sigma_0 \frac{\partial u}{\partial t} + \delta(x - x_P)\tilde{F} \tag{3.171}$$

where $\delta(x - x_P)$ is a Dirac delta function centred in x_P, so that attached element is localised in a single point of string, and again, the force is scaled, i.e. $\tilde{F} = \frac{F}{m_s}$, where m_s is the string mass. The force term is responsible for a type of preparation. It may assume the following form [61]

$$\tilde{F} = -\omega_0^2 u(x_P) - \omega_1^4 \left(u(x_P)\right)^3 - 2\sigma_P \frac{\partial u(x_P)}{\partial t} \tag{3.172}$$

where the first term represents a linear spring, the second – a cubic non-linear spring, and the last – a damping [61]. In such form it may be considered a model of wedged rubber.

This basic model may be approximated with the following finite difference scheme [61]

$$\delta_{tt} u_l[n] = \gamma^2 \delta_{xx} u_l[n] - 2\sigma_0 \delta_{t\cdot} u_l[n] + J_p(x_P)\tilde{F} \tag{3.173}$$

$$\begin{aligned} \tilde{F} = &- \omega_0^2 \mu_{t\cdot}\eta[n] - \omega_1^4 \eta[n]^2 \mu_{t\cdot}\eta[n] - 2\sigma_P \delta_{t\cdot}\eta[n] \\ \eta[n] = &\, I_p(x_P) u_l[n] \end{aligned} \tag{3.174}$$

where I_p is the p-th order interpolation operator, and J_p is the p-th order spreading operator. As Bilbao points out, the result of such system is strictly dissipative, even though the system is not linear [61]. It may not be clear how to solve a simultaneous dependence of (3.173) and (3.174) on $u[n+1]$. Bilbao proposes a solution shown in Figure 3.60.

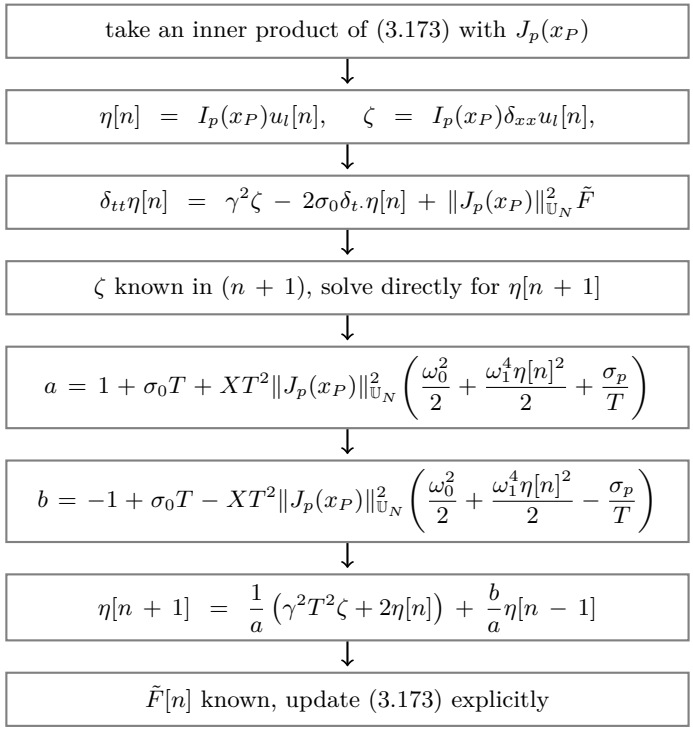


Figure 3.60. Method of solving a simultaneous dependence on $u[n+1]$ in (3.173) and (3.174)
Source: author's elaboration, based on Bilbao [61]

If one considers a simplified force term from (3.172) with $\omega_1 = 0$ and $\sigma_P = 0$, the CFL $\lambda < 1$ remains the stability condition for grid points other than l_P. In l_P a stronger condition is required [61]

$$\frac{X}{2}(1-\lambda^2) - \frac{T^2\omega_0^2}{8} > 0 \tag{3.175}$$

Further stability considerations can be found in works of Bilbao [61] and Rabenstein [453].

Another kind of preparation involves rattling elements. Beginning over with (3.171) the force term now can be written as [61]

$$\tilde{F} = \begin{cases} -\omega_R^{\alpha+1} \left(u(x_P) - u_R - \frac{1}{2}\epsilon\right)^{\alpha} & u(x_P) - u_R \geq \frac{1}{2}\epsilon \\ 0 & |u(x_P) - u_R| < \frac{1}{2}\epsilon \\ \omega_R^{\alpha+1} \left(u_R - u(x_P) - \frac{1}{2}\epsilon\right)^{\alpha} & u(x_P) - u_R \leq \frac{1}{2}\epsilon \end{cases} \tag{3.176}$$

$$\frac{\partial^2 u_R}{\partial t^2} = -\mathcal{M}\tilde{F} \tag{3.177}$$

where u_R is the distance between the centre of rattling element and the string rest position at location x_P, ϵ is the length of rattle, ω_R is the stiffness parameter, α is the stiffness exponent, and $\mathcal{M}$ is the mass ratio of string to rattle.

A linear string model does not reproduce an interesting effect that occurs under large amplitudes of vibration – a pitch glide, also referred to as **tension modulation** [579, 558, 183]. Large amplitudes can also be a source of another effect, referred to as **phantom partials** [129, 33]. The effect may be observed when coupling between transverse and longitudinal vibration causes instability in string motion, resulting in beating.

The simplest model of a string suitable for handling large amplitudes is the Kirchhoff–Carrier model [295, 101, 400, 17]

$$\rho A \frac{\partial^2 u}{\partial t^2} = \left(T_0 + \frac{EA}{2L} \int_0^L \left(\frac{\partial u}{\partial x} \right)^2 dx \right) \frac{\partial^2 u}{\partial x^2} \tag{3.178}$$

where ρ is the density of string material, T_0 is the string tension, E is the Young's modulus, A is the area of string cross-section, and L is the string length. The model has been utilised in the analysis of non-linear phenomena [163, 164, 272, 482], in musical acoustics [215, 331], and in sound synthesis [184, 62, 423]. Due to dependence of response on the amplitude, scaling of the system involves not only coordinates (3.110), but also the dependent variable

$$\tilde{u} = \frac{u}{L} \tag{3.179}$$

though as was the case of linear string, tilde symbol will be omitted in scaled system, which assumes the following form [61]

$$\frac{\partial^2 u}{\partial t^2} = \gamma^2 \mathfrak{G} \frac{\partial^2 u}{\partial x^2} \tag{3.180}$$

where

$$\mathfrak{G} = 1 + \frac{\alpha^2}{2} \left\| \frac{\partial u}{\partial x} \right\|_{\mathbb{U}}^2 \tag{3.181}$$

$$\gamma = \frac{1}{L}\sqrt{\frac{T_0}{\rho A}} \tag{3.182}$$

$$\alpha = \sqrt{\frac{EA}{T_0}} \tag{3.183}$$

Parameter α is responsible for the strength of stiffness in relation to string tension. Similarly to the linear string, a loss term can be added to the equation [61]

$$\frac{\partial^2 u}{\partial t^2} = \gamma^2 \mathfrak{G} \frac{\partial^2 u}{\partial x^2} - 2\sigma_0 \frac{\partial u}{\partial t} \tag{3.184}$$

Non-linearity depends on the amplitude, which is gradually decreased by loss. An auditory result of this behaviour is a downward pitch-glide.

The Kirchhoff–Carrier system with loss can be approximated with the following finite difference scheme [61]

$$\delta_{tt} u_l[n] = \gamma^2 \mathfrak{g} \delta_{xx} - 2\sigma_0 \delta_{t\cdot} u_l[n] \tag{3.185}$$

where a scalar time series $\mathfrak{g}$ is the approximation of $\mathfrak{G}$, and under the assumption of fixed boundary conditions it may be expressed in the explicit form [61]

$$\mathfrak{g} = 1 + \frac{1 + \dfrac{\alpha^2}{2} \left\| \delta_{x+} u_l[n] \right\|^2_{\underline{\mathbb{U}_N}}}{1 + \dfrac{\gamma^2 T^2 \alpha^2}{4} \left\| \delta_{xx} u_l[n] \right\|^2_{\underline{\mathbb{U}_N}}} \tag{3.186}$$

An underline or overline added to the domain symbol represents a removal of boundary points

$$\begin{aligned} \underline{\mathbb{U}_N} &= [0, 1, \ldots, N-1] \\ \overline{\mathbb{U}_N} &= [1, 2, \ldots, N] \\ \underline{\overline{\mathbb{U}_N}} &= [1, 2, \ldots, N-1] \end{aligned} \tag{3.187}$$

A behaviour of the Kirchhoff–Carrier system approximated using a scheme given by (3.185) and (3.186) without loss is presented in Figure 3.61. A pitch glide effect produced by the Kirchhoff–Carrier system with added linear loss term is presented in Figure 3.62.

Reproduction of the phantom partials effect requires a more complex model than the Kirchhoff–Carrier equation [32] – one that includes both longitudinal and transverse motion [391, 568]. In such model a growing excitation amplitude causes an increase in number and strength of inharmonic partials. The most prominent of these components are referred to as phantom partials [129]. Principles governing their actual frequencies were studied by Bank [32].

For the purpose of sound synthesis Bilbao proposes the following model [61]

$$\begin{aligned} \rho A \frac{\partial^2 u}{\partial t^2} &= EA \frac{\partial^2 u}{\partial x^2} - (EA - T_0) \frac{\partial}{\partial x} \left(\frac{\partial \Phi}{\partial q} \right) \\ \rho A \frac{\partial^2 \zeta}{\partial t^2} &= EA \frac{\partial^2 \zeta}{\partial x^2} - (EA - T_0) \frac{\partial}{\partial x} \left(\frac{\partial \Phi}{\partial p} \right) \end{aligned} \tag{3.188}$$

where $u(x,t)$ is the transverse string displacement, $\zeta(x,t)$ is the longitudinal string displacement, auxiliary variables p and q are defined as

$$p = \frac{\partial \zeta}{\partial x} \qquad q = \frac{\partial u}{\partial x} \tag{3.189}$$

and the function Φ, coupling both equations, is expressed as:

$$\Phi = \sqrt{(1+p)^2 + q^2} - 1 - p \tag{3.190}$$

The remaining parameters (ρ, A, E, and T_0) are the same as in the Kirchhoff–Carrier equation (3.178).

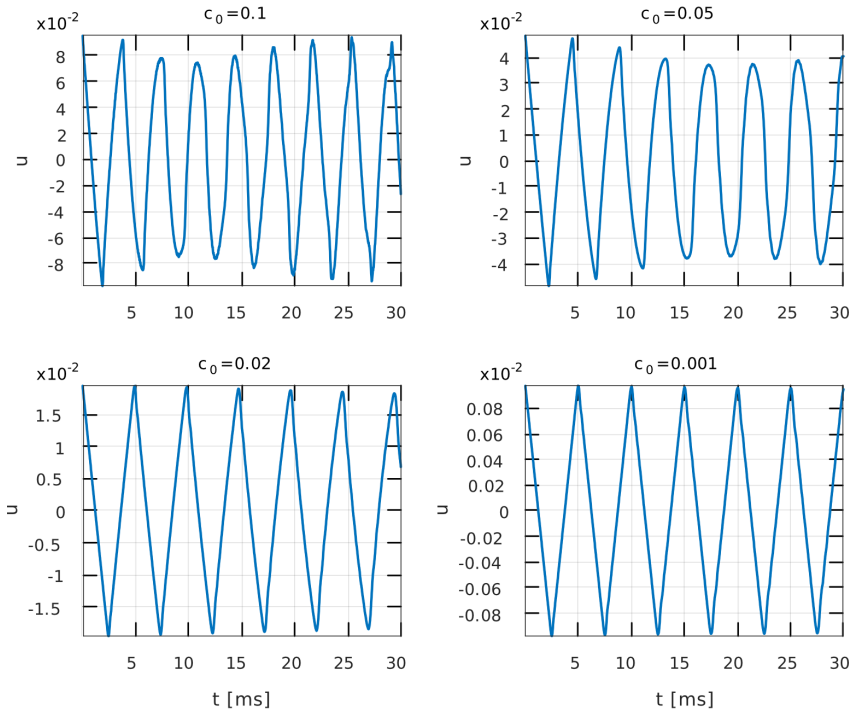


Figure 3.61. Signal produced by the Kirchhoff–Carrier model (3.184) approximated by (3.185) and (3.186) for different values of peak initial displacement c_0; simulation parameters were set to: $\gamma = 400$, $\alpha = 10$, $\lambda = 0.7$, and the initial displacement was triangular, centred at $x_c = 0.5$

Source: author's elaboration, based on Bilbao [61]

Using a scaling principle similar to the one applied in case of the Kirchhoff–Carrier equation, the system transforms into [61]

$$\begin{aligned}\frac{\partial^2 u}{\partial t^2} &= \gamma^2\alpha^2\frac{\partial^2 u}{\partial x^2} - \gamma^2(\alpha^2-1)\frac{\partial}{\partial x}\left(\frac{\partial \Phi}{\partial q}\right)\\ \frac{\partial^2 \zeta}{\partial t^2} &= \gamma^2\alpha^2\frac{\partial^2 \zeta}{\partial x^2} - \gamma^2(\alpha^2-1)\frac{\partial}{\partial x}\left(\frac{\partial \Phi}{\partial p}\right)\end{aligned} \tag{3.191}$$

Other, more general models can also be applied [400, 599, 311].

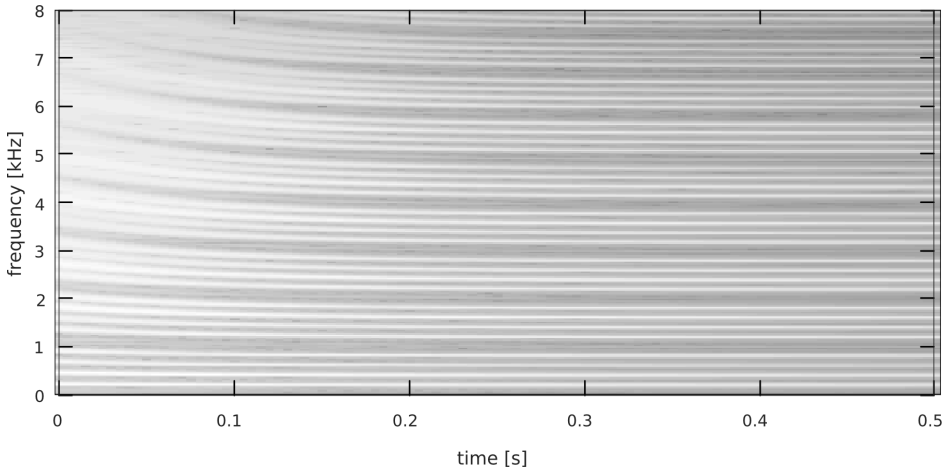


Figure 3.62. Spectrogram illustrating a pitch glide effect produced by the Kirchhoff–Carrier model with linear loss term (3.185) and (3.186); simulation parameters were set to: $\gamma = 400$, $\alpha = 10$, $\lambda = 0.7$, $T_{60} = 2$ s; the initial displacement was a plucked raised cosine, $x_c = 0.8$ and $x_{hw} = 0.1$
Source: author's elaboration, based on Bilbao [61]

Model from (3.191) requires a careful choices while designing an appropriate finite difference scheme, due to conflicting stability conditions. Bilbao proposes several approximations [61], though in order to obtain satisfactory results, without extensive dispersion, it is necessary to use distinct grids for longitudinal and transverse displacements, X_ζ and X_u respectively, and connect them through interpolation. The approximation assumes the following form [61]

$$\begin{aligned}\delta_{tt}u_l[n] &= \gamma^2\delta_{xx}u_l[n] + \gamma^2\frac{\alpha^2-1}{2}\delta_{x+}\left(q^2\mu_{t\cdot}q + 2q\mu_{tt}p\right)\\ \delta_{tt}\zeta_m[n] &= \gamma^2\alpha^2\delta_{xx}\zeta_m[n] + \gamma^2\frac{\alpha^2-1}{2}\delta_{x+}\mathcal{I}^*_{X_u\to X_\zeta,p}(q\mu_{t\cdot}q)\end{aligned} \tag{3.192}$$

where functions p and q are given as

$$p = \delta_{x-}\mathcal{I}_{X_\zeta\to X_u,p}\zeta \qquad q = \delta_{x-}u \tag{3.193}$$

Symbols $\mathcal{I}_{X_\zeta \to X_u, p}$ and $\mathcal{I}^*_{X_u \to X_\zeta, p}$ represent upsampling interpolant and its downsampling complex conjugate of order p, as defined in (3.97) and (3.98).

Models presented up to this point assume a transverse string motion to be planar. In real instruments the motion is more complex and in time can become non-planar, which leads to the effect of **whirling** [272, 421, 483], where the energy is transferred between two string polarisations. Modelling of the phenomenon requires representing the transverse displacement by a vector holding two components orthogonal to the string axis. The effect has been studied by Gough [215].

3.2.1.8. Bar

Vibrating elements of numerous musical instruments may be idealised as bars. Principles of bar vibrations have been extensively studied [391, 219, 191], and various models have been proposed [219, 403]. For instance, in cases of non-thin bars the linear Timoshenko model of beam vibration may be an appropriate choice, yet for sound synthesis purposes modelling may start with transverse vibrations of a basic, thin, ideal, uniform bar. Bilbao states, that such bar can be described in the simplest way with the Euler–Bernoulli model which in a lossless case assumes the following form [61]

$$\rho A \frac{\partial^2 u}{\partial t^2} = -EI \frac{\partial^4 u}{\partial x^4} \tag{3.194}$$

where ρ is the density of a bar material, A is the area of its cross-section, E is the Young's modulus, and I is the moment of inertia. In the ideal bar these quantities are constants.

For the convenience of implementation, the model can be scaled using the same principle that was applied in the case of a string (3.110), which results in the following form [61]

$$\frac{\partial^2 u}{\partial t^2} = -\kappa^2 \frac{\partial^4 u}{\partial x^4} \tag{3.195}$$

where all constants are combined in one stiffness parameter κ, defined as for the stiff string (3.132).

Contrary to wave equation, where the wavenumber scales directly with frequency, in case of bar the frequency scales with the square of the wavenumber. It is a cause of dispersion – components with larger wavenumber travel faster.

While it was sufficient for a string to assume fixed boundary conditions, a bar can be ended in various manners, leading to the following pairs of conditions [61]

$$\begin{aligned} u = \frac{\partial u}{\partial x} &= 0 \qquad \text{clamped} \\ u = \frac{\partial^2 u}{\partial x^2} &= 0 \qquad \text{simply supported} \\ \frac{\partial^2 u}{\partial x^2} = \frac{\partial^3 u}{\partial x^3} &= 0 \qquad \text{free} \end{aligned} \tag{3.196}$$

A simple explicit finite difference scheme for the ideal bar may be written as [61]

$$\delta_{tt} u_l[n] = -\kappa^2 \delta_{xxxx} u_l[n] \tag{3.197}$$

and in the form of recursion

$$\begin{aligned} u_l[n+1] = (2-6\mu^2)u_l[n] + 4\mu^2(u_{l+1}[n] + u_{l-1}[n]) - \\ - \mu^2(u_{l-2}[n] + u_{l+2}[n]) - u_l[n-1] \end{aligned} \tag{3.198}$$

where the parameter μ is defined as

$$\mu \triangleq \kappa \frac{T}{X^2} \tag{3.199}$$

The role of μ is similar to λ in the wave equation. It is used to define stability condition [61]

$$\mu \leq \frac{1}{2} \quad \longrightarrow \quad T \leq \frac{X^2}{2\kappa} \tag{3.200}$$

An implicit θ-scheme allows to maximise bandwidth of produced signal [61]

$$(\theta + (1-\theta)\mu_{x\cdot})\delta_{tt} u_l[n] = -\kappa^2 \delta_{xxxx} u_l[n] \tag{3.201}$$

and it has the following stability condition [61]

$$\theta \geq \frac{1}{2} \qquad \mu \leq \frac{\sqrt{2\theta - 1}}{2} \tag{3.202}$$

It is not uncommon in musical instruments that the assumption of uniform bar cross-section does not hold. One example is the marimba with arch-cut bars [191]. In such cases not only the cross-section area A, but also the moment of inertia I are functions of thickness, varying with x [61]

$$\rho A(x) \frac{\partial^2 u}{\partial t^2} = -\frac{\partial^2}{\partial x^2} \left(EI(x) \frac{\partial^2 u}{\partial x^2} \right) \tag{3.203}$$

If the cross-section is rectangular, then the area can be expressed as

$$A = bH_0 \phi(x) \tag{3.204}$$

and the moment of inertia as

$$I = \frac{1}{12} b H_0^3 \phi^3 \tag{3.205}$$

where b is the width of a bar, H_0 is the reference thickness, and $\phi(x)$ is the variation around H_0.

In scaled variables (3.203) assumes the following form

$$\phi \frac{\partial^2 u}{\partial t^2} = -\kappa_0^2 \frac{\partial^2}{\partial x^2} \left(\phi^3 \frac{\partial^2 u}{\partial x^2} \right) \tag{3.206}$$

where

$$\kappa_0^2 = \frac{EH_0^2}{12\rho L^4} \tag{3.207}$$

Bar described by (3.206) requires the following modification to the formulation of free boundary condition [61]

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial}{\partial x}\left(\phi^3 \frac{\partial^2 u}{\partial x^2}\right) = 0 \tag{3.208}$$

An application of a simple finite difference scheme based directly on (3.206) leads to extreme dispersion and serious decrease in bandwidth of produced signal for any but the smallest variations of cross-section area. Instead, Bilbao suggests to apply stretched coordinate $\alpha(x)$ accommodating cross-section variations [61]

$$\alpha(x) = \frac{1}{\alpha_{av}} \int_0^x \frac{1}{\sqrt{\phi(\eta)}} d\eta \tag{3.209}$$

where

$$\alpha_{av} = \int_0^1 \frac{1}{\sqrt{\phi(\eta)}} d\eta \tag{3.210}$$

In α coordinate (3.206) transforms into [61]

$$\phi^{\frac{3}{2}} \frac{\partial^2 u}{\partial t^2} = -\frac{\kappa_0^2}{\alpha_{av}^4} \frac{\partial}{\partial \alpha}\left(\phi^{-\frac{1}{2}} \frac{\partial}{\partial \alpha}\left(\phi^{\frac{5}{2}} \frac{\partial}{\partial \alpha}\left(\phi^{-\frac{1}{2}} \frac{\partial}{\partial \alpha}\right)\right)\right) \tag{3.211}$$

which allows to design the following finite difference scheme [61]

$$\left[\phi^{\frac{3}{2}}\right] \delta_{tt} u_l[n] = -\frac{\kappa_0^2}{\alpha_{av}^4} \delta_{\alpha+}\left(\mu_{\alpha-}\phi^{-\frac{1}{2}} \delta_{\alpha-}\left(\phi^{\frac{5}{2}} \delta_{\alpha+}\left(\mu_{\alpha-}\phi^{-\frac{1}{2}} \delta_{\alpha-} u_l[n]\right)\right)\right) \tag{3.212}$$

where $[\phi^{\frac{3}{2}}]$ is the discrete approximation to $\phi^{\frac{3}{2}}$. Bilbao gives the following stability condition for the scheme [61]

$$\mu = \frac{T\kappa_0}{X^2} \leq \frac{\alpha_{av}^2}{2} \min\left(\sqrt{\frac{\left[\phi^{\frac{3}{2}}\right]}{\mu_{\alpha+}\left(\left(\mu_{\alpha-}\phi^{\frac{5}{2}}\right)\left(\mu_{\alpha-}\phi^{-\frac{1}{2}}\right)^2\right)}}\right) \approx \frac{\alpha_{av}^2}{2} \tag{3.213}$$

As a further improvement, Chaigne and Doutaut propose an implicit scheme [108].

In order to simulate the pitch glide effect similar to that occurring in string, a nonlinear bar model is required [224]. In scaled form it is expressed as [61]

$$\frac{\partial^2 u}{\partial t^2} = -\kappa^2 \frac{\partial^4 u}{\partial x^4} + \frac{\gamma_l^2}{2} \left\| \frac{\partial u}{\partial x} \right\|_{\mathbb{U}}^2 \frac{\partial^2 u}{\partial x^2} - 2\sigma_0 \frac{\partial u}{\partial t} \tag{3.214}$$

where

$$\gamma_l^2 = \frac{E}{\rho L^2} \tag{3.215}$$

The system may be approximated using the following finite difference scheme [61]

$$\delta_{tt} u_l[n] = -\kappa^2 \delta_{xxxx} u_l[n] + \frac{\gamma_l^2}{2} \langle \delta_{x+} u_l[n], \mu_t . \delta_{x+} u_l[n] \rangle_{\underline{\mathbb{U}_N}} \delta_{xx} u_l[n] - 2\sigma_0 \delta_t . u_l[n] \tag{3.216}$$

3.2.1.9. Acoustic Tube

An acoustic tube is an enclosure with length in a single coordinate x significantly greater than in the others. Apart from the material properties of the enclosed air, a behaviour of such object depends also on the function $S(x)$ describing the area of its cross-section (Fig. 3.63).

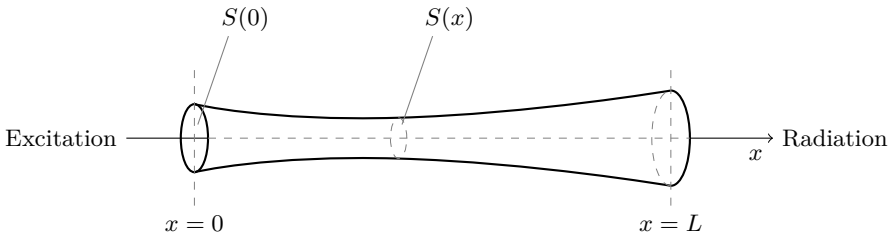


Figure 3.63. A 1D acoustic tube

The model can be based on linearisation of the equations of fluid dynamics [61]

$$\begin{aligned} \frac{S}{\rho c^2} \frac{\partial p}{\partial t} &= -\frac{\partial u}{\partial x} \\ \frac{\rho}{S} \frac{\partial u}{\partial t} &= -\frac{\partial p}{\partial x} \end{aligned} \tag{3.217}$$

where $p(x,t)$ is the pressure, $u(x,t)$ is the velocity, ρ is the density, and c is the wave speed. The actual model is referred to as Webster's equation [61]

$$S \frac{\partial^2 \Psi}{\partial t^2} = c^2 \frac{\partial}{\partial x} \left(S \frac{\partial \Psi}{\partial x} \right) \tag{3.218}$$

where

$$p = \rho \frac{\partial \Psi}{\partial t} \qquad u = -S \frac{\partial \Psi}{\partial x} \tag{3.219}$$

The model assumes linearity, as well as variations in u, p, and Ψ to be of a scale larger than the tube width [48]. An equivalent form of the Webster's equation with constant coefficient at the second spatial derivative can be written as [61]

$$\frac{\partial^2 \Phi}{\partial t^2} = c^2 \left(\frac{\partial^2 \Phi}{\partial x^2} - a(x) \Phi \right) \tag{3.220}$$

where

$$\Phi = \sqrt{S}\Psi \qquad a(x) = \frac{\dfrac{\partial^2 S}{\partial x^2}S - \dfrac{1}{2}\left(\dfrac{\partial S}{\partial x}\right)^2}{2S^2} \tag{3.221}$$

Scaling the Webster's equation involves scaling x according to (3.110), with the cross-section surface area given by

$$\tilde{S} = \frac{S(x)}{S_0} \tag{3.222}$$

where S_0 is the reference surface area. It is often convenient to assume it to be the area of the excitation end of a tube: $S_0 = S(0)$, which implies $\tilde{S}(0) = 1$. The dependent variables are scaled according to

$$\tilde{p} = \frac{p}{\rho c} \qquad \tilde{u} = \frac{u}{cS_0} \tag{3.223}$$

After skipping tildes, scaled system (3.217) assumes the following form [61]

$$\begin{gathered} S\frac{\partial p}{\partial t} = -\gamma\frac{\partial u}{\partial x} \\ \frac{1}{S}\frac{\partial u}{\partial t} = -\gamma\frac{\partial p}{\partial x} \\ x \in \mathbb{U} \end{gathered} \tag{3.224}$$

where γ is given by (3.111). With

$$\tilde{\Psi} = \frac{\Psi}{cL} \qquad \tilde{\Phi} = \frac{\Phi}{cL\sqrt{S_0}} \tag{3.225}$$

the Webster's equation can be scaled as well [61]

$$\begin{gathered} S\frac{\partial^2 \Psi}{\partial t^2} = \gamma^2 \frac{\partial}{\partial x}\left(S\frac{\partial \Psi}{\partial x}\right) \\ \frac{\partial^2 \Phi}{\partial t^2} = \gamma^2\left(\frac{\partial^2 \Phi}{\partial x^2} - a(x)\Phi\right) \\ x \in \mathbb{U} \end{gathered} \tag{3.226}$$

where

$$p = \frac{1}{\gamma}\frac{\partial \Psi}{\partial t} \qquad u = -S\frac{\partial \Psi}{\partial x} \tag{3.227}$$

As a convention in sound synthesis, left end of a tube, at $x = 0$, is regarded as closed one with excitation mechanism [61]. Therefore Neumann boundary condition may be applied

$$\frac{\partial}{\partial x}\Psi(0,t) = 0 \tag{3.228}$$

Right end is considered as radiating, and in the simplest approach it may be modelled as an open end, with Dirichlet boundary condition

$$\frac{\partial}{\partial t}\Psi(1,t) = 0 \tag{3.229}$$

In a more complex approach one may include the effect of inertia and loss, which brings the model closer to real instruments [61]

$$\frac{\partial}{\partial x}\Psi(1,t) = -\alpha_1 \frac{\partial}{\partial t}\Psi(1,t) - \alpha_2 \Psi(1,t) \tag{3.230}$$

Parameters α_1 and α_2 have to be adjusted to a particular tube. For a termination on an infinite plane they may be calculated according to the following formula [22]

$$\alpha_1 = \frac{1}{2(0.8216)^2\gamma} \qquad \alpha_2 = \frac{L}{0.8216\sqrt{S_0 S(1)\pi^{-1}}} \tag{3.231}$$

More details regarding boundary conditions for acoustic tubes along with expressions for α_1 and α_2 covering different types of tubes can be found in works of Rabiner and Schafer [456], and Bilbao [61].

Webster's equation may be approximated using the following finite difference scheme [581, 61]

$$[S]\delta_{tt}\Psi = \gamma^2 \delta_{x+}\left((\mu_{x-}S)(\delta_{x-}\Psi)\right) \tag{3.232}$$

where the grid function $[S]_l$ is an approximation to the continuous function $S(x)$, and the grid function S_l represents sampled values of the function $S(x)$. The scheme can be expanded as follows

$$\begin{aligned}\Psi_l[n+1] =& \frac{\lambda^2(S_{l+1}+S_l)}{2[S]_l}\Psi_{l+1}[n] + \frac{\lambda^2(S_l+S_{l-1})}{2[S]_l}\Psi_{l-1}[n] + \\ &+ \left(2 - \frac{\lambda^2(S_{l+1}+2S_l+S_{l-1})}{2[S]_l}\right)\Psi_l[n] - \Psi_l[n-1]\end{aligned} \tag{3.233}$$

3.2.1.10. Reed Excitation Mechanism

One of common excitation mechanisms in wind instruments is the reed. Model of reed and bore interaction has been discussed by McIntyre et al. [360], further investigated by Barjau et al. [34], and implemented in sound synthesis by Guillemain et al. [222, 223]. Other approaches have been proposed by Bilbao [59, 60], Avanzini and Rocchesso [25], and Dalmont et al. [148].

The single reed (Fig. 3.64) may be modelled as a lumped linear oscillator that is driven by drop of pressure across the mouthpiece. The equation of motion may be written as [61]

$$\frac{\mathrm{d}^2 y}{\mathrm{d}t^2} + 2\sigma_0 \frac{\mathrm{d}y}{\mathrm{d}t} + \omega_0^2(u - H_0) - \frac{\omega_1^{\alpha+1}}{H_0^{\alpha-1}}\left(|[y]^-|\right)^\alpha = -\frac{S_r p_\Delta}{M_r} \tag{3.234}$$

where y is the reed displacement in relation to equilibrium position H_0, M_r is the reed mass, S_r is the effective reed surface area, σ_0 is the damping parameter, ω_0 is the resonant frequency, ω_1 is responsible for the repelling force in the collision between reed and mouthpiece for $y < 0$, α is the power law non-linearity exponent, and $[y]^- = \frac{1}{2}(y - |y|)$.

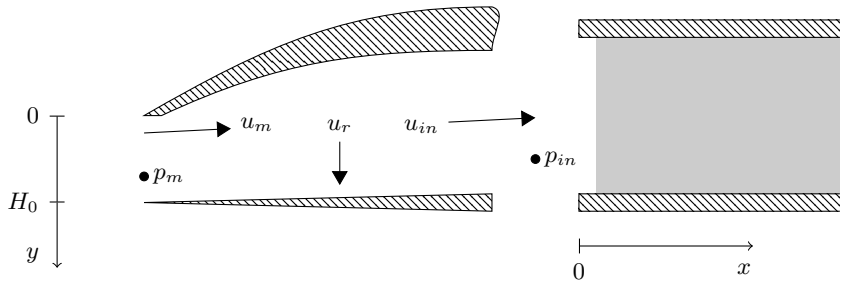


Figure 3.64. A single reed model
Source: author's elaboration, based on Bilbao [61]

Bernoulli's law relates the pressure difference across the mouthpiece [61]

$$p_\Delta = p_m - p_{in} \tag{3.235}$$

where p_m is the mouth pressure, and p_{in} is the pressure at the acoustic tube entrance, to the flow in the mouthpiece u_m [61]

$$u_m = w[y]^+ \sqrt{\frac{2|p_\Delta|}{\rho}} \operatorname{sgn}(p_\Delta) \tag{3.236}$$

where w is the reed channel width, and $[y]^+ = \frac{1}{2}(y + |y|)$. The flow is subject to a conservation law [61]

$$u_{in} = u_m - u_r \tag{3.237}$$

where u_{in} is the flow which enters the tube, and u_r depends on the reed displacement [61]

$$u_r = S_r \frac{\mathrm{d}y}{\mathrm{d}t} \tag{3.238}$$

The following scaling allows to couple reed model with an acoustic tube modelled by Webster's equation [61]

$$\tilde{y} = \frac{y}{H_0} - 1 \qquad \tilde{p} = \frac{p}{\rho c^2} \qquad \tilde{u} = \frac{u}{cS_0} \tag{3.239}$$

where u is any velocity variable, p is any pressure variable, c is the wave speed, ρ is the air density, and S_0 is the cross-section of left tube entrance. The reed model with scaled variables is described by the following system [61]

$$\begin{aligned} \frac{\mathrm{d}^2 y}{\mathrm{d}t^2} + 2\sigma_0 \frac{\mathrm{d}y}{\mathrm{d}t} + \omega_0^2 y - \omega_1^{\alpha+1} \left(|[y+1]^-|\right)^\alpha &= -\mathcal{Q} p_\Delta \\ p_\Delta &= p_m - p_{in} \\ u_m &= \mathcal{R}[y+1]^+ \sqrt{|p_\Delta|}\mathrm{sgn}(p_\Delta) \\ u_{in} &= u_m - u_r \\ u_r &= \mathcal{S}\frac{\mathrm{d}y}{\mathrm{d}t} \end{aligned} \tag{3.240}$$

where

$$\mathcal{Q} = \frac{\rho c^2 S_r}{M_r H_0} \qquad \mathcal{R} = \sqrt{2}\frac{w H_0}{S_0} \qquad \mathcal{S} = \frac{S_r H_0}{c S_0} \tag{3.241}$$

The following semi-implicit finite difference scheme may be used to approximate system (3.240) [61]

$$\begin{aligned} &\delta_{tt} y[n] + 2\sigma_0 \delta_{t\cdot} y[n] + \omega_0^2 \mu_{t\cdot} y[n] - \\ &- \omega_1^{\alpha+1} \left(\mu_{t\cdot}(y[n]+1)\right) \left(|[y[n]+1]^-|\right)^{\alpha-1} = -\mathcal{Q} p_\Delta \end{aligned} \tag{3.242}$$

and

$$u_r = \mathcal{S}\delta_{t\cdot} y[n] \tag{3.243}$$

The remaining members of the system do not include time differentiation, therefore they can be calculated directly. Connection between the reed model and the Webster's equation in $x = 0$ can be performed in the following manner [61]

$$p_{in} = \frac{1}{\gamma}\delta_{t\cdot}\Psi_0 \qquad u_{in} = -\delta_{x\cdot}\Psi_0 \tag{3.244}$$

Despite complexity of the system coupled with Webster's equation, Bilbao presents the implementation that allows to update it explicitly [61] through appropriately chosen order of computations and application of selected difference operator identities.

In the lumped reed model mouth pressure p_m is usually regarded as user-controlled external data source, preferably coming from some form of physical controller. Its value has an impact not only on the signal amplitude, but also on timbre due to the effect referred to as reed beating [290, 586, 61], originating from collision between the reed and the mouthpiece. Equilibrium displacement H_0 can also be user-controlled, and related to embouchure force. If its rate of change is slow, and so is the rate of change of parameters given in (3.241), the model (3.240) can be still applied. By experimenting with various input parameter settings even such simple model is able to produce not only a pitched sound, but also a set of various effects, such as squeak, warble, or even sounds similar to multiphonics [28]. The one-mass model however, is too simple to reproduce reed bending. Such behaviour requires a simulation to be carried out using distributed reed models [26].

3.2.1.11. Toneholes in Acoustic Tube

The basic mechanism of controlling pitch in woodwind instruments is a set of toneholes which can be left open, stay closed, or be partially closed. Toneholes may be considered branched side-tubes [61], and in the simplest approach they are defined by two parameters: the radius b and the height τ (Fig. 3.65). Fundamental research on toneholes was carried out by Benade [45, 46]. Tonehole is usually modelled through analogy to electrical N-port, as an impedance or scattering matrix that links pressures and volume velocities [287, 288, 289, 404, 405, 173]. Such models were applied in methods based on digital waveguides and wave digital filters [575, 491, 527, 587]. A tonehole can also be modelled as a lumped element with mass and stiffness, directly in time domain [174]. Another approach involves multiple convolutions with reflection functions from toneholes [356] and feedback to the reed [35].

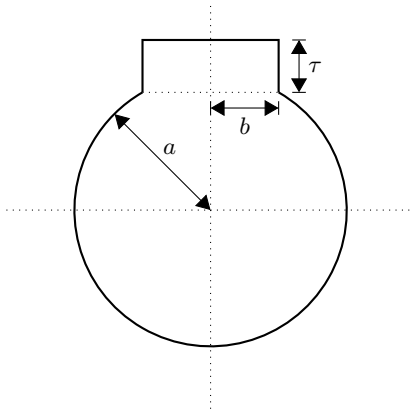


Figure 3.65. A cross-section of a woodwind instrument bore with a tonehole; τ is the height of the tonehole, b is the radius of the tonehole, and a is the radius of the instrument bore cross-section area
Source: author's elaboration, based on Bilbao [61]

A basic circuit representation of the tonehole is the two-port with impedances $\frac{1}{2}Z_a(s)$ and $Z_s(s)$ on series branches and on shunt branch, respectively. The remaining variables include Laplace transforms of pressures on the left and right side of the tonehole, $\hat{p}_-$ and $\hat{p}_+$, and Laplace transforms of volume velocities, $\hat{u}_-$ and $\hat{u}_+$. All variables are considered scaled and dimensionless.

Pressure and velocity drops can be calculated as [61]

$$\hat{p}_{\text{diff}} = \hat{p}_+ - \hat{p}_- = -\epsilon(x)\frac{Z_a}{2}(\hat{u}_+ + \hat{u}_-) \tag{3.245}$$

$$\hat{u}_{\text{diff}} = \hat{u}_+ - \hat{u}_- = -\epsilon(x)\frac{1}{2Z_s + \frac{1}{2}Z_a}(\hat{p}_+ + \hat{p}_-) \tag{3.246}$$

where $\epsilon(x)$ is a distribution with unit area and peak in the tonehole location. Assuming spatial continuity of pressures and velocities, (3.245) and (3.246) can be simplified to [61]

$$\hat{p}_{\text{diff}} = -\epsilon(x) Z_a \hat{u} \tag{3.247}$$

$$\hat{u}_{\text{diff}} = -\epsilon(x) \frac{1}{Z_s + \frac{1}{4} Z_a} \hat{p} \tag{3.248}$$

Both drop values can be used in system (3.224) after applying Laplace transformation [61]

$$\begin{aligned} Ss\hat{p} &= -\gamma \frac{\partial \hat{u}}{\partial x} - \frac{\gamma \epsilon(x)}{Z_s + \frac{1}{4} Z_a} \hat{p} \\ \frac{s}{S} \hat{u} &= -\gamma \frac{\partial \hat{p}}{\partial x} - \gamma \epsilon(x) Z_a \hat{u} \end{aligned} \tag{3.249}$$

Impedances Z_a and Z_s can be calculated according to Keefe's tonehole model [288]

$$Z_a(s) = -\frac{s \xi_a^{o,c}}{\gamma S_T} \quad \text{open and closed tonehole} \tag{3.250}$$

$$Z_s = \begin{cases} \dfrac{1}{S_T} \coth\left(\dfrac{s\xi}{\gamma}\right) & \text{closed tonehole} \\ \dfrac{s\xi_e}{\gamma S_T} & \text{open tonehole} \end{cases} \tag{3.251}$$

where S_T is the ratio of the area of tonehole cross-section to the area of bore cross-section at its left end, and ξ is the ratio of the tonehole height τ to the length of tube L [288, 289, 587]. The model lacks certain details, such as dependence of effective length on the frequency or loss term, which are discussed by Keefe [288], Ducasse [174], or Bilbao [61].

Z_a can be substituted to the second expression in (3.249), and the result can be transformed back to the time domain. The effect is the following partial differential equation [61]

$$\frac{1}{S^*} \frac{\partial u}{\partial t} = -\gamma \frac{\partial p}{\partial x} \quad \text{where} \quad S^*(x) = \frac{S(x)}{1 - \dfrac{S(x) \xi_a \epsilon(x)}{S_T}} \tag{3.252}$$

The area function $S^*(x)$ equals $S(x)$ everywhere except near the peak of $\epsilon(x)$, i.e. in the surroundings of the tonehole. (3.252) can be solved exactly by choosing velocity potential Ψ, such as that [61]

$$u = -S^* \frac{\partial \Psi}{\partial x} \qquad p = \frac{1}{\gamma} \frac{\partial \Psi}{\partial t} \tag{3.253}$$

Series impedance Z_s may be approximated as [61]

$$Z_s = \begin{cases} \dfrac{\gamma}{s\xi S_T} & \text{closed tonehole} \\ \dfrac{s\xi_e}{\gamma S_T} & \text{open tonehole} \end{cases} \tag{3.254}$$

Closed tonehole behaves like a stiffness, and open – like a mass. The first expression in (3.249) can now be transformed into [61]

$$Ss\hat{p} = -\gamma\frac{\partial\hat{u}}{\partial x} - \frac{\gamma\epsilon(x)}{a_1 s + \dfrac{a_2}{s}}\hat{p} \tag{3.255}$$

where

$$a_1 = \begin{cases} -\dfrac{\xi_a^c}{4\gamma S_T} & \text{closed} \\ \dfrac{\xi_e - \frac{1}{4}\xi_a^c}{\gamma S_T} & \text{open} \end{cases} \qquad a_2 = \begin{cases} \dfrac{\gamma}{\xi S_T} & \text{closed} \\ 0 & \text{open} \end{cases} \tag{3.256}$$

With the use of (3.253), after an inverse Laplace transformation, a coupled system that describes a connection between a tonehole and a bore can be formulated [61]

$$\begin{aligned} S\frac{\partial^2\Psi}{\partial t^2} &= \gamma^2\frac{\partial}{\partial x}\left(S^*\frac{\partial\Psi}{\partial x}\right) - \epsilon(x)\frac{\mathrm{d}m}{\mathrm{d}t} \\ a_1\frac{\mathrm{d}^2 m}{\mathrm{d}t^2} + a_2 m &= \gamma\left\langle\frac{\partial\Psi}{\partial t}, \epsilon\right\rangle_{\mathbb{U}} \end{aligned} \tag{3.257}$$

Function $m(t)$ stores energy in the lumped element.

A more convenient for implementation purposes, yet greatly simplified tonehole model, has been used by van Walstijn and Scavone [491, 587]. In this approach $Z_s = 0$, and both, open and closed toneholes are handled together by combining mass and stiffness in parallel

$$Z_s = \left(\frac{\phi\gamma S_T}{s\xi_e} + \frac{(1-\phi)s\xi S_T}{\gamma}\right)^{-1} \tag{3.258}$$

where $0 \leq \phi \leq 1$ is the parameter that controls opening ($\phi = 1$) and closing ($\phi = 0$) of the tonehole. Obviously it can take intermediate values as well, and gradually change over time to reproduce various fingering effects, including multiphonics and note transitions. Without $Z_a(s)$ the area function $S^* = S$. If the $\epsilon(x)$ distribution is approximated with Dirac delta function $\delta(x - x_T)$ centred at the tonehole location x_T, the Webster's equation can be written as [61]

$$\begin{aligned} S\frac{\partial^2\Psi}{\partial t^2} &= \gamma^2\frac{\partial}{\partial x}\left(S\frac{\partial\Psi}{\partial x}\right) - \delta(x - x_T)m \\ m &= \frac{\phi\gamma^2 S_T}{\xi_e}\Psi + (1-\phi)\xi S_T\frac{\partial^2\Psi}{\partial t^2} \end{aligned} \tag{3.259}$$

A finite difference approximation for the system given by (3.259) has been proposed by Bilbao [61]

$$\begin{aligned}\mu_{xx} S \delta_{tt} \Psi =& \gamma^2 \delta_{x+} \left((\mu_{x-} S) \delta_{x-} \Psi \right) - J_p(x_T) m \\ m =& \frac{\phi \gamma^2 S_T}{\xi_e} \left(\alpha I_p(x_T) \Psi + (1-\alpha) \mu_t . I_p(x_T) \Psi \right) + \\ &+ (1-\phi) \xi S_T \delta_{tt} I_p(x_T) \Psi\end{aligned} \tag{3.260}$$

where interpolation ($I_p(x_T)$) and spreading ($J_p(x_T)$) operators allow to position the tonehole at any chosen location, and parameter α allows to control the behaviour of the simulation. The stability condition is [61]

$$\lambda \leq 1 \qquad \alpha \leq \frac{1}{2} \tag{3.261}$$

3.2.1.12. Other Wind Instruments

Single-reed instruments have been studied and simulated more frequently than other wind instruments, though a number of studies involve double-reed and brass instruments, as well as flutes and recorders. Double-reed mechanisms require models that are better at handling non-linear flow effects [12, 591, 222, 11]. In brass instruments a valve formed by player's lips opens at high pressures, unlike in reed instruments, where a reed valve is shut by high pressure [47, 191]. Synthesis of brass sounds can be carried out in a manner similar to reed instruments [2, 478, 593, 594]. In some cases however, high blowing pressures resulting in shock waves may require models of non-linear wave propagation [237, 394, 592]. Horns with wider flares may also require going beyond one-dimensional models [407]. Turbulent flow makes synthesis of flute and recorder sounds more difficult, and models applied in sound synthesis are either significantly simplified [360, 127, 574, 589, 572, 590], or are far from operation in real time [3].

3.2.1.13. Membrane

The simplest model of a vibrating **membrane** is the wave equation in two dimensions

$$\frac{\partial^2 u}{\partial t^2} = c^2 \Delta u \tag{3.262}$$

where c is the wave speed, and Δ is the Laplacian operator as in (3.69) or (3.70). Scaling of 2D wave equation is carried out similarly to 1D case, with γ given by (3.111)

$$\frac{\partial^2 u}{\partial t^2} = \gamma^2 \Delta u \tag{3.263}$$

though the characteristic length L has to be chosen differently. One possible choice for Cartesian coordinates is to set it to [61]

$$L = \sqrt{|\mathcal{D}|} \tag{3.264}$$

With such choice of L, (3.263) is defined over a region of unit area. For simulations in polar coordinates a convenient choice is to set $L = R$ [61].

A basic, ideal initial conditions are similar to the case of 1D wave equation, i.e.

$$\begin{aligned} u(x, y, 0) &= u_0(x, y) \qquad \text{(pluck)} \\ \frac{\partial}{\partial t} u(x, y, 0) &= v_0(x, y) \qquad \text{(strike)} \end{aligned} \tag{3.265}$$

Both, u_0 and v_0, may be modelled with the raised cosine distribution, which in 2D assumes the following form [61]

$$c_{rc}(x, y) = \begin{cases} \frac{c_0}{2} \left(1 + \cos\left(\frac{\pi r}{r_{hw}}\right)\right) & r \leq r_{hw} \\ 0 & r > r_{hw} \end{cases} \tag{3.266}$$

where $r = \sqrt{(x - x_0)^2 + (y - y_0)^2}$, c_0 is the peak amplitude, r_{hw} is the half-width, and (x_0, y_0) is the peak position.

The most basic finite difference approximation for the 2D wave equation, in Cartesian coordinates with Laplacian approximated by a five-point operator (3.72), can be written as [61]

$$\delta_{tt} u_{l,m}[n] = \gamma^2 \delta_{\Delta\boxplus} u_{l,m}[n] \tag{3.267}$$

The scheme expands to the following recursion

$$\begin{aligned} u_{l,m}[n+1] = &\lambda^2 \left(u_{l+1,m}[n] + u_{l-1,m}[n] + u_{l,m+1}[n] + u_{l,m-1}[n]\right) + \\ &+ 2(1 - 2\lambda^2) u_{l,m}[n] - u_{l,m}[n-1] \end{aligned} \tag{3.268}$$

Assuming equal grid spacing in both spatial dimensions, i.e. $X_x = X_y$, the Courant number λ, similarly to the 1D case, is defined as in (3.115). Stability condition for the scheme is [61]

$$\lambda \leq \frac{1}{\sqrt{2}} \tag{3.269}$$

At the limit ($\lambda = 2^{-\frac{1}{2}}$), (3.268) simplifies by losing term with $u_{l,m}[n]$.

Scheme (3.267) has the following first-order accurate boundary conditions (for the case of lower boundaries)

$$\begin{aligned} u_{0,m\geq 0} &= 0 \qquad & u_{l\geq 0,0} &= 0 \qquad & \text{(Dirichlet)} \\ \delta_{x-} u_{0,m\geq 0} &= 0 \qquad & \delta_{x-} u_{l\geq 0,0} &= 0 \qquad & \text{(Neumann)} \end{aligned} \tag{3.270}$$

Second-order accurate conditions can be formulated using centred difference operators.

As an alternative, one can utilise a parametrised scheme [61] by approximating Laplacian with nine-point operator (3.74)

$$\delta_{tt} u_{l,m}[n] = \gamma^2 \delta_{\Delta\alpha} u_{l,m}[n] = \gamma^2 \left(\alpha \delta_{\Delta\boxplus} + (1 - \alpha) \delta_{\Delta\boxtimes}\right) u_{l,m}[n] \tag{3.271}$$

For $\alpha = 1$ (3.271) reduces to (3.267). Stability condition is given by [61]

$$\alpha \geq 0 \qquad \lambda \leq \min\left(1, \frac{1}{\sqrt{2\alpha}}\right) \tag{3.272}$$

Parameter α allows to control dispersion characteristics of the scheme – examples with various settings can be found in the work of Bilbao [61].

The third variant of the approximation is a compact implicit scheme [61], expressed as

$$\delta_{tt} u_{l,m}[n] = \gamma^2 \left(1 + \frac{T^2(1-\theta)}{2}\delta_{tt}\right) \delta_{\Delta\alpha} u_{l,m}[n] \tag{3.273}$$

This scheme has two parameters: α from nine-point Laplacian approximation, and the additional one – θ. Stability condition is given by [61]

$$\alpha \geq 0 \quad \begin{cases} \lambda \leq \sqrt{\dfrac{\min\left(1, \dfrac{1}{2\alpha}\right)}{\sqrt{2\theta - 1}}} & \theta > \dfrac{1}{2} \\ \lambda \text{ unconstrained} & \theta \leq \dfrac{1}{2} \end{cases} \tag{3.274}$$

while θ is not constrained. By adjusting parameter values, the implicit scheme allows for much closer match to the ideal phase velocity [61]. More schemes for 2D wave equation can be found in works of Bilbao, Strikwerda, and Kowalczyk [59, 538, 307].

The most common membranes in percussion instruments are circular, therefore it is useful to consider the 2D wave equation in polar coordinates. If the equation is defined over the unit circle, i.e. a circle with radius $R = 1$, and scaling is as follows

$$\gamma = \frac{c}{R} \tag{3.275}$$

then the scaled 2D wave equation in polar coordinates assumes the following form [61]

$$\frac{\partial^2 u}{\partial t^2} = \gamma^2 \left(\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial u}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2}\right) \tag{3.276}$$

The simplest explicit finite difference scheme for (3.276) is given by [61]

$$\delta_{tt} u_{l,m}[n] = \gamma^2 \delta_{\Delta\circ} u_{l,m}[n] \tag{3.277}$$

where $\delta_{\Delta\circ}$ is the operator approximating Laplacian as in (3.76). However, such explicit scheme is not particularly well suited for sound synthesis purposes, unless applied at extremely high sampling frequency. As Bilbao points out [61], its frequency bandwidth is severely limited, and simulation results display extreme numerical dispersion. Better results can be obtained by applying a parametrised implicit scheme [61]

$$(1 + \gamma^2 T^2 \alpha \delta_{\Delta\circ}) \delta_{tt} u_{l,m}[n] = \gamma^2 \delta_{\Delta\circ} u_{l,m}[n] \tag{3.278}$$

Scheme (3.278) has the following stability condition [61]

$$\frac{4\left(\alpha+\frac{1}{4}\right)\gamma^2T^2}{X_r^2}\left(1+\frac{1}{X_\theta^2}\right)\leq 1 \tag{3.279}$$

which is required for $\alpha \geq -\frac{1}{4}$. For other values of the parameter the scheme is unconditionally stable.

3.2.1.14. Plate

2D wave equation may be a basis for a simple model of a membrane. A plate has an inherent stiffness, therefore it requires another equation as a model basis. An uniform, thin, isotropic plate can be simulated using the Kirchhoff model [391]

$$\rho H\frac{\partial^2 u}{\partial t^2}=-D\Delta\Delta u \tag{3.280}$$

where ρ is the plate material density, and H is the plate thickness. Parameter D, sometimes referred to as flexural rigidity, is defined as [61]

$$D=\frac{EH^3}{12(1-\nu^2)} \tag{3.281}$$

where E is the Young's modulus, and $\nu<\frac{1}{2}$ is the Poisson's ratio.

After scaling similar to one applied in the case of a membrane, the Kirchhoff plate model changes to [61]

$$\frac{\partial^2 u}{\partial t^2}=-\kappa^2\Delta\Delta u \tag{3.282}$$

where

$$\kappa^2=\frac{D}{\rho HL^4} \tag{3.283}$$

Clamped and simply supported boundary conditions can be adapted directly from the bar model (3.196), though lossless free boundary condition is different, depends on ν, and assumes the following form [61]

$$\frac{\partial^2 u}{\partial x^2}+\nu\frac{\partial^2 u}{\partial y^2}=\frac{\partial^3 u}{\partial x^3}+(2-\nu)\frac{\partial^3 u}{\partial x\partial y^2}=0 \tag{3.284}$$

An additional corner condition emerges if a plate is free at a corner [61]

$$\frac{\partial^2 u}{\partial x\partial y}|_{x=0,y=0}=0 \tag{3.285}$$

Various finite difference schemes for rectangular plates can be found in the work of Szilard [544]. One of the most basic schemes for the Kirchhoff plate can be written as [61]

$$\delta_{tt}u_{l,m}[n]=-\kappa^2\delta_{\Delta\boxplus,\Delta\boxplus}u_{l,m}[n] \tag{3.286}$$

If grid spacing is equal in both dimensions, i.e. $X_x = X_y = X$, the scheme can be rewritten as the following recursion

$$\begin{aligned} u_{l,m}[n+1] = (2 - 20\mu^2)u_{l,m}[n] + \\ + 8\mu^2(u_{l,m+1}[n] + u_{l,m-1}[n] + u_{l+1,m}[n] + u_{l-1,m}[n]) - \\ - 2\mu^2(u_{l+1,m+1}[n] + u_{l+1,m-1}[n] + u_{l-1,m+1}[n] + u_{l-1,m-1}[n]) - \\ - \mu^2(u_{l,m+2}[n] + u_{l,m-2}[n] + u_{l+2,m}[n] + u_{l-2,m}[n]) - u_{l,m}[n-1] \end{aligned} \tag{3.287}$$

where

$$\mu = \frac{\kappa T}{X^2} \tag{3.288}$$

The scheme is stable for $\mu \leq \frac{1}{4}$ [61], which is more conveniently written as

$$X \geq 2\sqrt{\kappa T} \tag{3.289}$$

The scheme (3.286) has the following lossless numerical boundary conditions

$$\begin{aligned} u_{0,m} = \delta_{x-} u_{0,m} = 0 & \qquad \text{clamped} \\ u_{0,m} = \delta_{xx} u_{0,m} = 0 & \qquad \text{simply supported} \\ (\delta_{xx} + \nu\delta_{yy})u_{0,m} = \delta_{x-}\left(\delta_{xx} + (2-\nu)\delta_{yy}\right) u_{0,m} = 0 & \qquad \text{free} \\ \delta_{x-y-} u_{0,0} = 0 & \qquad \text{corner, free} \end{aligned} \tag{3.290}$$

Due to audible dispersion effects introduced by scheme (3.286), higher-pitched plates may be modelled more accurately with an implicit scheme instead. An example of such scheme, with two parameters, α and ϕ, is given by Bilbao [61]

$$\left(1 + \alpha T\kappa\delta_{\Delta\boxplus} + \phi T^2\kappa^2\delta_{\Delta\boxplus,\Delta\boxplus}\right)\delta_{tt}u_{l,m}[n] = -\kappa^2\delta_{\Delta\boxplus,\Delta\boxplus}u_{l,m}[n] \tag{3.291}$$

As was the case of a string, a plate model may be refined by including terms responsible for loss and tension. A basic variant of such model assumes the form as presented hereunder [61]

$$\frac{\partial^2 u}{\partial t^2} = -\kappa^2\Delta\Delta u + \gamma^2\Delta u - 2\sigma_0\frac{\partial u}{\partial t} + 2\sigma_1\Delta\frac{\partial u}{\partial t} \tag{3.292}$$

and its explicit approximation is given by the following scheme [61]

$$\begin{aligned} \delta_{tt}u_{l,m}[n] = & -\kappa^2\delta_{\Delta\boxplus,\Delta\boxplus}u_{l,m}[n] + \gamma^2\delta_{\Delta\boxplus}u_{l,m}[n] - \\ & - 2\sigma_0\delta_{t\cdot}u_{l,m}[n] + 2\sigma_1\delta_{t-}\delta_{\Delta\boxplus}u_{l,m}[n] \end{aligned} \tag{3.293}$$

A plate can be excited in a way similar to string excitation, i.e. by a mallet or a bow. The excitation is added to the model through inclusion of force term with a spatial distribution. Such model for a plate with loss, but with no tension, may be written as [61]

$$\frac{\partial^2 u}{\partial t^2} = -\kappa^2\Delta\Delta u - 2\sigma_0\frac{\partial u}{\partial t} + 2\sigma_1\Delta\frac{\partial u}{\partial t} + \epsilon(x,y)\tilde{F} \tag{3.294}$$

where $\epsilon(x,t)$ is the force spatial distribution, for instance, the Dirac delta or raised cosine, and $\tilde{F} = \tilde{F}(t)$ is the force divided by the mass of a plate. A simple example of a finite difference scheme that approximates (3.294) may assume the following form [61]

$$\begin{aligned}\delta_{tt} u_{l,m}[n] = &- \kappa^2 \delta_{\Delta\boxplus,\Delta\boxplus} u_{l,m}[n] - 2\sigma_0 \delta_{t\cdot} u_{l,m}[n] + \\ &+ 2\sigma_1 \delta_{t-} \delta_{\Delta\boxplus} u_{l,m}[n] + J_p(x_i, y_i)\tilde{F}\end{aligned} \tag{3.295}$$

where (x_i, y_i) is the excitation position, and J_p is the p-th order spreading operator. The scheme is explicit due to use of backward difference operator δ_{t-} in connection with Laplacian approximation. It can be changed to centred difference operator at a cost of making the scheme implicit.

A mallet-like object striking a plate [109, 314], or a membrane [461, 313], may be modelled with the following force term [61]

$$\tilde{F} = -\mathcal{M}\frac{\mathrm{d}^2 u_M}{\mathrm{d}t^2} = \omega_M^{1+\alpha} \left([u_M - \langle \epsilon, u \rangle_{\mathcal{D}}]^+ \right)^{\alpha} \tag{3.296}$$

where u_M is the vertical position of the mallet, $\mathcal{M}$ is the mallet to plate mass ratio, ω_M is the mallet stiffness parameter, α is the stiffness exponent, and $[\cdot]^+$ operation is defined as in (3.153). Bilbao proposes to approximate the model with the following semi-implicit scheme [61]

$$\begin{aligned}&\tilde{F} = -\mathcal{M}\delta_{tt} u_M[n] = \\ &= \omega_M^{1+\alpha} \mu_{t\cdot} \left(u_M[n] - I_p(x_i, y_i) u_{l,m}[n] \right) \left([u_M - I_p(x_i, y_i) u_{l,m}[n]]^+ \right)^{\alpha-1}\end{aligned} \tag{3.297}$$

where I_p is the p-th order interpolation operator. In a simpler, although slightly less realistic approach, there is no coupling between models of plate an mallet. The excitation function can be specified directly in the scheme, e.g. in the form of a pulse [61]

$$\tilde{F}(t) = \begin{cases} \dfrac{\tilde{F}_{\max}}{2} \left(1 - \cos\left(\dfrac{2\pi(t - t_0)}{T_{\mathrm{exc}}} \right) \right) & t_0 \leq t \leq t_0 + T_{\mathrm{exc}} \\ 0 & \text{otherwise} \end{cases} \tag{3.298}$$

where $\tilde{F}_{\max}$ is the peak force, T_{exc} is the duration of pulse, and t_0 is the start time of the pulse.

Circular plates in percussion instruments can be modelled in polar coordinates. The base equation is the same as for the Cartesian coordinates (3.282), however the bi-Laplacian operator is different (3.78), and scaling leads to a different expression for parameter κ

$$\kappa^2 = \frac{D}{\rho H R^4} \tag{3.299}$$

where R is the radius of the circular plate. The boundary conditions are given by [61]

$$\begin{aligned} u = \frac{\partial u}{\partial r} = 0 \quad &\text{clamped} \\ u = \frac{\partial^2 u}{\partial r^2} + \nu\frac{\partial u}{\partial r} + \nu\frac{\partial^2 u}{\partial \theta^2} = 0 \quad &\text{simply supported} \\ \frac{\partial^2 u}{\partial r^2} + \nu\frac{\partial u}{\partial r} + \nu\frac{\partial^2 u}{\partial \theta^2} = & \\ = \frac{\partial^3 u}{\partial r^3} + \frac{\partial^2 u}{\partial r^2} - \frac{\partial u}{\partial r} + (\nu-3)\frac{\partial^2 u}{\partial \theta^2} + (2-\nu)\frac{\partial^3 u}{\partial r \partial \theta^2} = 0 \quad &\text{free} \end{aligned} \tag{3.300}$$

and if a plate is clamped within a small ϵ distance from its centre, as in some cymbals, additional condition emerges [61]

$$u = \frac{\partial u}{\partial r} = 0 \quad \text{at} \quad r = \epsilon \tag{3.301}$$

Due to high numerical dispersion and reduced bandwidth of explicit finite difference schemes for plates simulated in polar coordinates, Bilbao advises the use of a parametrised implicit one instead [61]

$$(1 + \alpha\kappa^2 T^2 \delta_{\Delta\circ,\Delta\circ})\delta_{tt} u_{l,m}[n] = -\kappa^2 \delta_{\Delta\circ,\Delta\circ} u_{l,m}[n] \tag{3.302}$$

The first-order accurate conservative boundary conditions for the free edge can be expressed as [61]

$$\begin{aligned} \left((\mu_{r+} r)\delta_{\Delta\circ} - (1-\nu)\delta_{r+} - (1-\nu)\delta_{\theta\theta}\right) u_{N_r-1,m} = 0 \\ \left((\mu_{r+} r)\delta_{r+}\delta_{\Delta\circ} + (1-\nu)\delta_{r+}\delta_{\theta\theta} - (1-\nu)\delta_{\theta\theta}\right) u_{N_r-1,m} = 0 \end{aligned} \tag{3.303}$$

where N_r is the number of grid points in r dimension, and corresponding grid range is $l \in [0, N_r - 1]$. Second-order accurate conditions can be found in the work of Bilbao [61], and shall be implemented for high values of κ. Additional clamped centre condition may be assumed as [61]

$$u_{0,0} = u_{1,m} = 0 \qquad \text{for all } m \tag{3.304}$$

The scheme is unconditionally stable for $\alpha > \frac{1}{4}$, but for other values stability depends on the centre point update scheme. Hence the conditions assume the following common form [61]

$$\frac{2\sqrt{1-4\alpha}\kappa T}{X_r^2}\left(1 + \frac{1}{C h_\theta^2}\right) \leq 1, \tag{3.305}$$

where $C = 1$ for the free, and $C = 4$ for the clamped centre.

High amplitudes of plate vibrations cause non-linear behaviour and characteristic auditory effects [479]. Some of the effects, such as pitch glide, can be reproduced with a relatively simple model, like the Berger's equation [54, 460], which has been exploited in sound synthesis [433]. Other effects however, for instance generation of

subharmonics or growth of energy in high-frequency range, require more elaborate models. One of such models is the plate model of von Kármán [403, 397]

$$\rho H \frac{\partial^2 u}{\partial t^2} = -D\Delta\Delta u + \mathcal{L}(\Phi, u) \qquad \Delta\Delta\Phi = -\frac{EH}{2}\mathcal{L}(u, u) \tag{3.306}$$

where $\Phi(x, y, t)$ is the Airy stress function, and the non-linear operator $\mathcal{L}$ in Cartesian coordinates, working on two functions, $u(x, y)$ and $v(x, y)$, is defined as [466]

$$\mathcal{L}(u, v) = \frac{\partial^2 u}{\partial x^2}\frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 u}{\partial y^2}\frac{\partial^2 v}{\partial x^2} - 2\frac{\partial^2 u}{\partial x \partial y}\frac{\partial^2 v}{\partial x \partial y} \tag{3.307}$$

Scaling (3.306) involves not only spatial coordinates, i.e. $\tilde{x} = \frac{x}{L}$, $\tilde{y} = \frac{y}{L}$, $L = \sqrt{|\mathcal{D}|}$, but also removing dimensionality from u and Φ, i.e. $\tilde{u} = \frac{u}{u_0}$ and $\tilde{\Phi} = \frac{\Phi}{\Phi_0}$. For the synthesis purposes coefficients u_0 and Φ_0 may be chosen so as to reduce the number of parameters in equations

$$\Phi_0 = D \qquad u_0 = \frac{H}{\sqrt{6(1-\nu^2)}} \tag{3.308}$$

Such choice leads to the following scaled form of the von Kármán plate model equation [61]

$$\frac{\partial^2 u}{\partial t^2} = -\kappa^2\Delta\Delta u + \kappa^2\mathcal{L}(\Phi, u) \qquad \Delta\Delta\Phi = -\mathcal{L}(u, u) \tag{3.309}$$

Discussion on boundary conditions for the model (3.309), particularly concerning non-obvious conditions for Φ, can be found in works of Bilbao [61], Muradova [397], Ilanko [261], Chien [114], and Geveci [205]. Some interpretations can be also found in publications of Schaeffer [493] and Horn [245]. Lossless conditions from the linear plate can be applied, however Φ requires a separate set of two conditions on the edge. The following set may be utilised for clamped and simply supported boundary [61]

$$\Phi = \frac{\partial \Phi}{\partial x} = 0 \tag{3.310}$$

in connection with appropriate condition for u. It may also be utilised for the free condition, although in this case it is more common to use higher derivatives of Φ [61].

Excitation and loss in the von Kármán model are introduced similarly to the linear case [61]

$$\begin{aligned} \frac{\partial^2 u}{\partial t^2} &= -\kappa^2\Delta\Delta u + \kappa^2\mathcal{L}(\Phi, u) - 2\sigma_0\frac{\partial u}{\partial t} + 2\sigma_1\Delta\frac{\partial u}{\partial t} + \epsilon\tilde{F}(t) \\ \Delta\Delta\Phi &= -\mathcal{L}(u, u) \end{aligned} \tag{3.311}$$

One of many possible finite difference schemes for the model (3.311) assumes the following form [61]

$$\begin{aligned} \delta_{tt}u_{l,m}[n] = &-\kappa^2\delta_{\Delta\boxplus,\Delta\boxplus}u_{l,m}[n] + \kappa^2\left[\mathcal{L}(\Phi_{l,m}[n], u_{l,m}[n]\right] - \\ &- 2\sigma_0\delta_{t\cdot}u_{l,m}[n] + 2\sigma_1\delta_{\Delta\boxplus}\delta_{t-}u_{l,m}[n] + \epsilon\tilde{F} \\ \delta_{\Delta\boxplus,\Delta\boxplus}\Phi = &-\left[\mathcal{L}(u_{l,m}[n], u_{l,m}[n])\right] \end{aligned} \tag{3.312}$$

where $[\mathcal{L}]$ is an unspecified second-order accurate approximation to the non-linear term. One of its possible approximations is given by [61]

$$\mathfrak{l}(u,v) = \delta_{xx}u\delta_{yy}v + \delta_{yy}u\delta_{xx}v - 2\mu_{x-}\mu_{y-}(\delta_{x+y+}u\delta_{x+y+}v) \tag{3.313}$$

Assuming $[\mathcal{L}] = \mathfrak{l}$, the scheme requires to solve a linear system for the second equation, but the first one is updated explicitly.

Further improvements of plate models may include simulation of plate curvature. Shell models are discussed in detail by Leissa [332], however instead of finite difference schemes a better solution for such cases is the finite element method. Difference schemes may be applied in cases of simple spherical curvatures, as an extension to the von Kármán model [550].

3.2.2. Networks of Lumped Elements

Lumped models employ networks of elements characterised by lumped parameters. The elements include masses connected by springs and damping elements, also referred to as dashpots. Networks can be formed into strings, membranes or other kinds of structures. The freedom of structure design makes lumped networks particularly well-suited for quasi-physical synthesis. A network can propagate a wave, and the sound output is produced by time integration. Such approach may be considered appropriate when dimensions of a modelled physical object are small compared to wavelength of a vibration, which imposes an upper-frequency and physical size validity limits for a model in question [526].

Initial attempts to apply the principle of ideal masses, springs, and dampers, connected into a network as a means to synthesize sound, were carried out by Cadoz et al. [96]. A development of CORDIS and CORDIS-ANIMA synthesis environments followed [95, 545], the latter intended for modelling of objects in two and three dimensions. The systems were among first large attempts at real-time physical modelling synthesis. They advanced over time, and the latest versions apart from producing sound, can visualise movements of lumped elements, and provide haptic feedback.

3.2.2.1. Lumped Elements

Method implemented in CORDIS considers three basic lumped elements [96, 557]. Inertial behaviour of elements is modelled on the basis of Newton's laws, using ordinary differential equations. An element of mass is described by

$$F = m\frac{\partial^2 x}{\partial t^2} \tag{3.314}$$

A spring is given by

$$F_1 = F_2 = -K(x_1 - x_2) \tag{3.315}$$

Finally, a damper, or an element of impedance, is described as follows

$$F_1 = F_2 = -Z\left(\frac{\partial x_1}{\partial t} - \frac{\partial x_2}{\partial t}\right) \tag{3.316}$$

where F is the force that drives the mass, F_1 and F_2 are the forces at both ends of the spring or the damper, K is the spring coefficient, and Z is the friction coefficient.

In a discrete form, with derivatives approximated by first order backward differences, (3.314) is given by [557]

$$F[n] = m(x[n] - 2x[n-1] + x[n-2]) \tag{3.317}$$

(3.315) is approximated by [557]

$$F_1[n] = F_2[n] = -K(x_1[n] - x_2[n]) \tag{3.318}$$

and (3.316) transforms into [557]

$$F_1[n] = F_2[n] = -Z(x_1[n] - x_1[n-1] - x_2[n] + x_2[n-1]) \tag{3.319}$$

A parallel connection of a spring and damper, which may represent non-linear contact forces, is referred to as the conditional link, defined as

$$F_1 = F_2 = -K(x_1 - x_2) - Z\left(\frac{\partial x_1}{\partial t} - \frac{\partial x_2}{\partial t}\right) \tag{3.320}$$

Its discrete approximation assumes the following form [557]

$$\begin{aligned} F_1[n] =& F_2[n] = -K(x_1[n] - x_2[n]) - \\ & - Z(x_1[n] - x_1[n-1] - x_2[n] + x_2[n-1]) \end{aligned} \tag{3.321}$$

3.2.2.2. Operation

Time integration of a working network is performed with an audio sampling frequency, or greater, not unlike in the finite difference method. The entire model is relatively simple. For instance, a network of N masses connected through $N-1$ links is capable of reproducing N harmonics [557]. Among other physical modelling methods, networks of lumped elements may be less accurate, yet more flexible in terms of defining various vibrating structures, not necessary having a physical counterpart.

3.2.3. Modal Synthesis

A formulation of the modal synthesis method originates from IRCAM [7, 6]. The method is based on the assumption that a sound-producing object may be modelled as a set of coupled, vibrating substructures, such as bodies, bridges, acoustic tubes, membranes, plates, or bells, described by modal data [557]. Substructures respond to excitations, and their coupled connections allow the energy to be distributed throughout the whole set. It is possible to apply the method to model arbitrarily complex structures, yet rapidly increasing computational cost makes simulation of more elaborate models impractical. Modal synthesis has been implemented first in the MOSAIC system [390], and later in Modalys software [180]. It is still actively developed [65, 618, 87].

3.2.3.1. Model Data

There are two sets of data that describe modelled substructure [557, 61]. The first set contains modal shapes, frequencies, and damping coefficients of resonant modes. The second one is the excitation data, such as initial conditions, locations and functions of excitations, or readout locations. Modal data, which is the solution of an eigenvalue problem, is obtained prior to the actual synthesis process, in either analytical or experimental way. The analytical way involves a PDE system that has to consider geometry, material properties, and boundary conditions. The experimental way, involving measurements with excitation and pickup devices, may be applied in case of more complex structures, where accurate analysis would be impossible. Modal data is stored in the *shape matrix* that describes relative displacements of structure points, where columns represent contributions of particular modes to a displacement. A single mode may be considered a second-order resonator, and all modes work in parallel [557].

3.2.3.2. Synthesis Process

With the shape matrix established, the actual synthesis process is carried out using initial conditions and the excitation data. For a structure consisting of N points the instantaneous velocity of a k-th point is given by [6]

$$\frac{\partial u_{k,n+1}}{\partial t} = \sum_{m=1}^{N} \Phi_{k,m} \frac{\sum_{l=1}^{P} \Phi_{l,i} F_{\text{ext},l,n+1} + \frac{\partial \phi_{m,t}}{\partial t} \frac{1}{T} - \omega_m^2 \phi_{m,t}}{\frac{1}{T} + 2\omega_m \xi_m + \omega_m^2 \Delta t} \tag{3.322}$$

where $\Phi_{k,m}$ is the contribution of the m-th mode to the displacement of k-th structure point, $F_{\text{ext},l,n+1}$ is the external force on point l, T is the time step, and m-th mode data is represented by the angular velocity ω_m, damping coefficient ξ_m, and instantaneous deflection ϕ_m. Thus the synthesis process relies on an evolution of weighted combination of modal functions.

In case of wind instruments, where flows need to be calculated, external forces $F_{\text{ext},l,n+1}$ are replaced with external flows $U_{\text{ext},l,n+1}$, and the result is multiplied by the air density ρ_0

$$p_{k,t+1} = \rho_0 \frac{\partial u_{k,n+1}}{\partial t} \tag{3.323}$$

External forces can originate from initial excitation data, and thus be known, but some of them may originate from coupling between structures, often non-linear. For instance, an interaction between the reed and the air column may be expressed as [6]

$$\begin{aligned} U_{\text{ext},0,t+1} &= B \sqrt[3]{(p_{m,t+1} - p_{0,t+1})^2 \xi^4} + S_0 \xi_{t+1} && \text{open reed} \\ U_{\text{ext},0,t+1} &= 0 \qquad \xi_{t+1} = 0 && \text{closed reed} \end{aligned} \tag{3.324}$$

where $U_{\text{ext},0}$ is the flow entering the bore, p_m and p_0 are the pressure values at the mouth and the bore, respectively, ξ is the reed position, B is the Backus constant, and $S_0\xi$ is the additional flow related to the reed displacement.

Number of equations required depends on the modelled acoustical system and its division into substructures. For instance, a tube with a reed mouthpiece and M toneholes, with substructures interacting on the basis of flow conservation, requires $(M + 1)$ equations (3.322).

3.2.3.3. Output

A sound output at a given time is obtained by projecting modal functions through inner products onto an observation state – the simplest one is a delta function [61]. The output waveform is generated in time domain. Individual modes are described by scalar second-order ODEs, and solution is obtained using time-integration techniques. Modal synthesis may be considered a numerical method for a diagonalised linear problem [61], therefore a state cannot be directly observed, unlike in direct time-domain methods. In modal synthesis the observation is carried out through reversal of the diagonalisation, i.e. the projection. As a consequence, a straightforward way to implement variable location of excitation or readout, which allowed to produce interesting effects in direct time-domain method, is not available in modal method.

3.2.4. Karplus–Strong Synthesis

Inclusion of Karplus–Strong (KS) synthesis into a group of physical modelling methods may be arguable. Roads [470] points to its principle based on a digital delay line, connecting it to the waveguide method, which is undoubtedly physical. Bilbao [61] or Tolonen et al. [557], contrarily, assign it either within abstract, or wavetable methods, respectively, due to lack of immediate physical interpretation offered for the computations performed. Regardless, here it is discussed among physical methods, as a direct predecessor, or a very early, incomplete formulation of a waveguide synthesis. Correspondence to a physical object has been later provided by Jaffe and Smith [264].

Karplus–Strong synthesis [286] is a method of producing sounds that closely resemble a plucked string. The sound production mechanism is based on a digital delay line in a feedback configuration, with signal modified in each pass of the loop, as shown in Figure 3.66. The delay line is implemented as a wavetable, although it does not store a fixed waveform. In a typical configuration wavetable data is initially random. During synthesis it is recirculated, and terminated with a low order digital filter, usually low-pass. In the most basic implementation the filter simply averages successive samples. The filter causes the evolution of a signal produced, imitating the evolution of a plucked string sound, with spectrally rich, bright, impulse-like beginning, and gradually decaying harmonics, finally ending with a single tone. Tolonen et al. [557] examine the frequency response of the KS algorithm and point out, that it may be considered a comb filter.

In view of properties of a signal produced, Karplus–Strong is an extremely efficient method. The operations involved in signal production are additions and multiplications required by a digital filter, and shifting of wavetable data. If the latter is implemented as a circular buffer, only a single variable pointing to the instantaneous beginning of a data needs to be shifted.

Filling the wavetable with random values has an interesting side-effect. If the initial data is not a copy of previous excitation, but is actually randomised for each note, the timbre and evolution will subtly differ from note to note. It is uncommon in simple digital methods, yet typical for acoustic or analogue instruments.

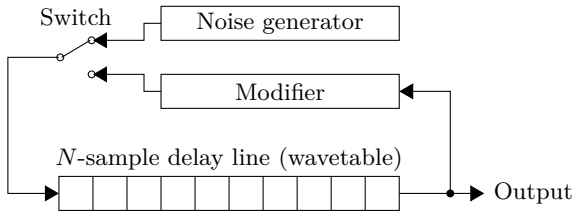


Figure 3.66. A general principle of Karplus–Strong algorithm for synthesis of a plucked string; after the wavetable has been initialised with random values, it is recirculated and modified

3.2.4.1. Basic Control

Looped operation and fixed length of the delay line result in a periodic signal. Its fundamental frequency is determined by the wavetable length N and sampling frequency f_s

$$f_0 = \frac{f_s}{N} \tag{3.325}$$

The same parameter N controls the decay time. Smaller values make contents of the wavetable being filtered more frequently, thus shortening the decay.

3.2.4.2. Plucked Strings and Drums

Figure 3.67 presents two basic variants of KS method. The first one produces sounds resembling plucked strings [557], and is given by the following expression

$$u[n] = \frac{1}{2}(u[n-N] + u[n-N-1]) \tag{3.326}$$

where N is the length of a wavetable. Drum-like sounds can be produced using the second configuration, that introduces random probability $0 \leq b \leq 1$, referred to as the *blend factor* [557]

$$u[n] = \begin{cases} \frac{1}{2}(u[n-N] + u[n-N-1]) & r < b \\ -\frac{1}{2}(u[n-N] + u[n-N-1]) & r \geq b \end{cases} \tag{3.327}$$

where $0 \leq r \leq 1$ is a random number with uniform distribution. If $b = 1$ the second configuration is reduced to the first one. Drum-like sounds are produced if $b = \frac{1}{2}$. In this case signal is aperiodic, and N controls its decay time only. A resonant drum

can be simulated by initialising the wavetable with a constant value instead of noise. Finally, setting $b = 0$ results in signal negation every $N + \frac{1}{2}$ samples. In effect, pitch is shifted an octave downwards, and spectrum has only odd harmonics [470].

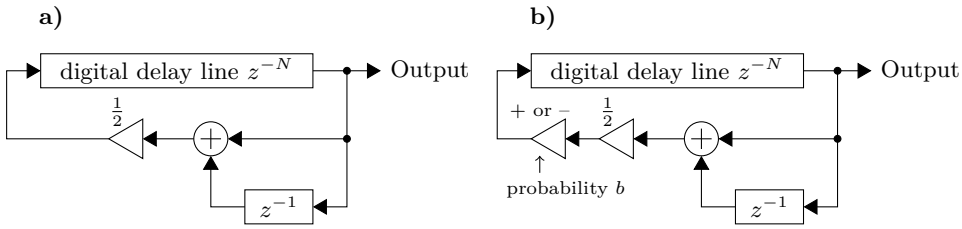


Figure 3.67. Two basic variants of the Karplus–Strong method: a) plucked string; b) drum-like instrument
Source: author's elaboration, based on Roads [470] and Tolonen et al. [557]

3.2.4.3. Decay Stretching

In both configurations decay time is controlled by table length N. This relation can be decoupled using the *decay stretching* technique [470]. The output signal is produced according to the following formula

$$u[n] = \begin{cases} u[n-N] & r < \left(1 - \frac{1}{s}\right) \\ \frac{1}{2}(u[n-N] + u[n-N-1]) & r < \frac{1}{s} \end{cases} \tag{3.328}$$

where s is the *stretch factor*. For $s = 1$ normal averaging is performed, and decay time is determined by N. However, decreasing s allows some number of samples not to be averaged, allowing to compensate for decay shortening. For values close to zero, very long sounds can be produced.

3.2.5. Waveguide Synthesis

Waveguide synthesis, initially formulated by Smith [520, 522, 524], is a method based on digital filter design and scattering theory [526]. Its basic form is one of the most efficient solutions among approaches based on physical modelling. The initial design was aimed at modelling instruments with one-dimensional vibrating element, which includes stringed, woodwinds, and many brass instruments. The model relies on a lossless wave equation (3.108) in a one-dimensional, linear, homogeneous medium, with lumped excitation elements, i.e. bows, hammers, or reeds, connected through scattering junctions in power conserving matrix operations [61].

3.2.5.1. Digital Waveguide

The key concept relies upon the d'Alembert's solution to the wave equation [557], with a linear combination of two non-interacting waves travelling left and right:

$u_L(x - ct)$ and $u_R(x + ct)$. The solution is implemented as a digital waveguide with discrete variables

$$x \to x_l = lX \qquad t \to t_n = nT \tag{3.329}$$

related by

$$c = \frac{X}{T} \tag{3.330}$$

where X is the spatial step and T is the time step. One can define

$$u^+[n] = u_R(nT) \qquad u^-[n] = u_L(nT) \tag{3.331}$$

which allows to write the solution in the following form [557]

$$\begin{aligned} u(x_l, t_n) &= u_R(t_n - \frac{x_l}{c}) + u_L(t_n + \frac{x_l}{c}) = \\ &= u_R(T(n-l)) + u_L(T(n+l)) = u^+[n-l] + u^-[n+l] \end{aligned} \tag{3.332}$$

The waveguide, shown in Figure 3.68, consists of two l-sample digital delay lines $u^+[n-l]$ and $u^-[n+l]$. Its physical output is obtained by summing a chosen point of both delay lines. By applying **fractional delay filters** [569, 312] it is possible to obtain values associated with non-integer positions. Waveguide operation requires no arithmetic, only shifts, thus the process is very efficient.

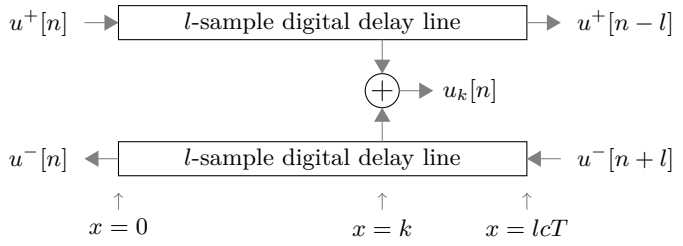


Figure 3.68. A digital waveguide with an observation point in k
Source: author's elaboration, based on Smith [522]

As Bilbao points out [61], waveguide synthesis has been inspired by Karplus–Strong method, and became a fully-fledged physical modelling method after two conceptual advancements: the association between a wavetable index and a position on a physical medium, and showing that data propagated through delay lines behave like travelling wave solutions to the wave equation, with their sum producing a physical quantity, i.e. displacement or pressure.

A subset of waveguide models is based on a different delay line configuration. It is referred to as a **single delay loop**, and it shares several features with Karplus–Strong method. As the name suggests, it is based on a single delay line, and on combining transfer functions of various model elements. The latter is valid due to

linearity and time-invariance of the waveguide. In comparison to a conventional pair of delay lines, the length of a single delay line is doubled, so that a total length of a looped part remains unchanged. Above-mentioned operations may be accompanied by further simplifications of the model, such as consolidating losses and discarding elements that would produce less salient auditory effects.

3.2.5.2. Dispersion, Damping, and Other Effects

Even though equation (3.332) represents only a lossless propagation, the waveguide method is able to simulate various effects through application of more elaborate filter blocks [61]. Two of commonly simulated phenomena are frequency-dependent damping, and dispersion [526].

Damping can be implemented by introducing a frequency-dependent gain factor $G(\omega)$ to attenuate travelling waves [522, 526]. For the sake of efficiency, and due to linear, time-invariant character of the waveguide, gain does not have to be introduced between every unit delay. It can be consolidated into $G^k(\omega)$, and inserted before the observation point.

Implementation of dispersion involves introducing an all-pass filter $H_k(\omega)$ before the observation point. Such filter approximates the effect of dispersion for a given length of the delay line, and can be efficiently designed as a series of one-pole all-pass filters [583]. Other design methods, leading to more accurate instrument simulations, have been considered as well [315, 1].

Another category of effects is related to pitch. For instance, Lagrange interpolation allows to perform fine-tuning of pitch and produce the effect of *glissando* [284]. Similar effect can be reproduced by applying all-pass fractional delay filters [569]. Välimäki, Tolonen, Karjalainen, and Erkut [579, 558, 183] developed techniques allowing to produce a pitch glide effect caused by non-linear behaviour, referred to as *tension modulation*.

A basic 1D wave equation produces periodic, harmonic signals, while acoustic instruments often manifest some degree of inharmonicity, which is both, a pitch and timbre related effect. A considerable degree of inharmonicity can be handled by applying banded waveguides [186, 187].

An attempt at reproducing timbre of acoustic instruments can be carried out by supplementing a model of vibrating element with a model of instrument body. It is efficiently applied in the **commuted waveguide synthesis** (CWS) [523, 285]. The method relies on division of sound production process into excitation, vibration of e.g. a string, and radiation by a body. All three can be interpreted as linear transfer functions excited by an impulse. The output is obtained by convolving respective impulse responses. Application of linear filters allows to commute the elements, and convolve excitation with body into a single impulse response. A number of such responses, representing various excitation characteristics, can be stored in wavetables, and applied to excite a string, with regards to a specific context [557].

3.2.5.3. Scattering Junction

More complex model configurations require connections between multiple waveguides, inputs and outputs. Such kind of operation is performed by scattering junc-

tions that connect N pairs of inputs and outputs with given impedances R_i. Values of wave variables v_i in the junction point are equal

$$v_S = v_1 = \ldots = v_N \tag{3.333}$$

and the sum of string forces or tube flows is zero

$$\sum_{k=1}^{N} f_k = 0 \tag{3.334}$$

Considering two travelling waves

$$\begin{aligned} v_k &= v_k^+ + v_k^- \qquad & f_k &= f_k^+ + f_k^- \\ f_k^+ &= R_k v_k^+ \qquad & f_k^- &= -R_k v_k^- \end{aligned} \tag{3.335}$$

one can obtain wave value in the junction point according to

$$v_S = \frac{2 \sum\limits_{k=1}^{N} R_k v_k^+}{\sum\limits_{k=1}^{N} R_k} \tag{3.336}$$

and

$$v_k^- = v_S - v_k^+ \tag{3.337}$$

3.2.5.4. Examples of Waveguide Configurations

First implementations of waveguide synthesis modelled strings, and the basic waveguide configuration for simulating a string vibration is presented in Figure 3.69. A pair of waveguides is separated with a scattering junction used to connect an excitation mechanism, such as a hammer or a plectrum. Both ends are terminated with digital filters that model boundary terminations, coupling with resonator, or with other strings. Output is read as a sum of values at the same position in both delay lines.

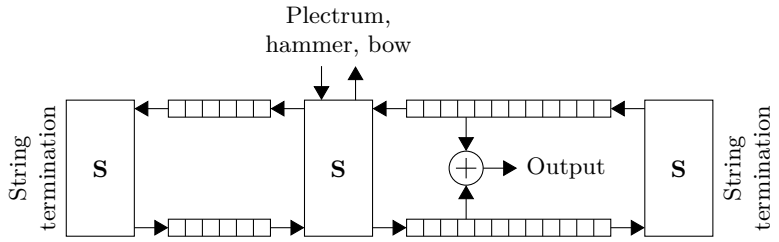


Figure 3.69. A basic waveguide model of a string; **S** represents a scattering operation, and small boxes represent individual unit delays
Source: author's elaboration, based on Bilbao [61]

A typical configuration for woodwinds or brass, shown in Figure 3.70, involves a waveguide that is broken up by scattering junctions representing toneholes. One end of the waveguide is connected to the excitation mechanism, i.e. lip or reed model. Output is produced on the other end, after a filter has been applied to model bell and radiation effects. Conical bores can be modelled as well, for instance, by dividing them into multiple waveguide segments representing different cross-section areas, referred to by Roads as *sampling in space* [470].

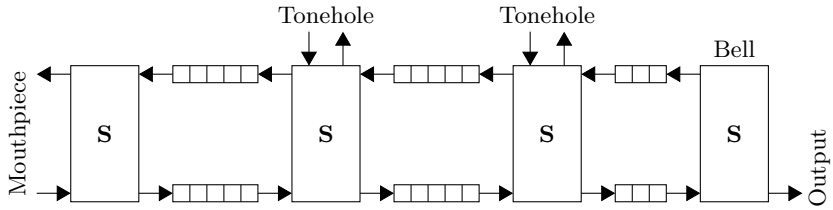


Figure 3.70. A basic waveguide model of a woodwind instrument; **S** represents a scattering operation, and small boxes represent individual unit delays
Source: author's elaboration, based on Bilbao [61]

A regular 2D or 3D network of digital waveguides is referred to as a **waveguide mesh**. A 2D variant (Fig. 3.71), with bi-directional delay units between 4-port scattering junctions, may simulate a membrane, while 3D structures can be applied to model room acoustics. Similarly to FD simulations, wave propagation in regular waveguide meshes is dispersive, thus propagation speed and magnitude response depend on both, the direction and the frequency. The effect can be attenuated by modifying the waveguide formulation [557].

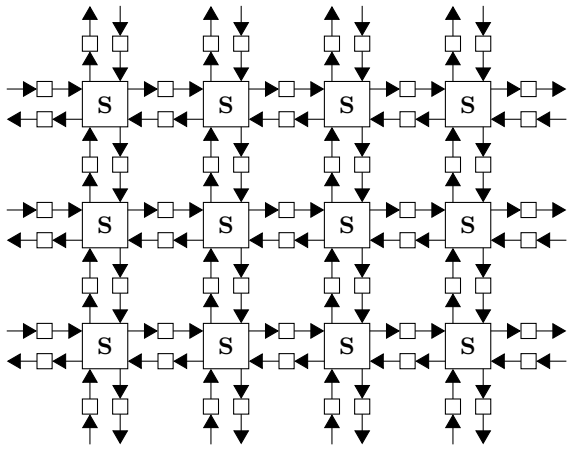


Figure 3.71. A regular waveguide mesh for modelling a two dimensional membrane; **S** represents a scattering operation, and small boxes represent individual unit delays
Source: author's elaboration, based on Bilbao [61]

3.2.5.5. Applications

Early applications of the waveguide synthesis aimed at modelling acoustic plucked strings. In addition to guitar [322, 323], other models included harpsichord [578], kantele [184, 427], or guqin [428]. Some attempts have been made to simulate electric guitar with distortion and feedback [542] as well as slapbass [459]. Models of struck string include piano [528, 585] and clavichord [571]. Finally, introduction of waveguide mesh allowed to simulate membranes [313].

Other than strings and membranes, models of woodwinds and brass have been developed. The former include clarinet [519, 576, 238], flute [574] and organ pipe [145], while the latter aim at simulating trombone [132, 394] and trumpet [165, 593, 592]. A relatively large group of models considers vocal and singing synthesis [131, 133, 395].

3.2.6. Other Physical Modelling Methods

It is possible to point physical modelling methods that are particularly convenient, realistic, or efficient when applied to simulate specific instruments, or – more precisely – certain elements of instruments. Therefore attempts have been made to combine advantages of various modelling techniques in what is generally referred to as **hybrid methods** [61]. Such attempts usually employ a modular approach [281, 577, 63] that allow user of a synthesizer to interconnect elements of choice, thus designing a compound instrument. Hybrids may combine objects simulated using various scattering techniques, including filters and waveguides, with direct numerical simulations, such as FD model approximations of lumped elements [584, 282, 283], and modal methods [432, 453].

While simulations based on the finite difference method are the most common in sound synthesis, time domain simulations can be performed using other methods as well. A notable example, widely applied in mechanical engineering, is the **finite element method** (FEM) [68, 137, 602, 143]. Regarding simulations of musical instruments, its key advantage is the ability to model complex geometric details. As of disadvantages, compared to FD, FEM requires considerably more effort to establish a simulation domain and divide it into elements (Fig. 3.72) in a way that would allow to take full advantage of the method's geometric modelling capabilities. FEM has been successfully applied to model instruments for research purposes [461, 30], although its sound synthesis applications are scarce so far [211].

Progress in computer technology facilitates advancement of numerical modelling methods. Established methods are further developed, and new methods emerge. Even though sound synthesis, and particularly its real-time, user controlled variant, imposes fairly strict conditions on the usability of a model, which needs to be extremely efficient computationally, yet sufficiently realistic and convincing with regards to auditory details, some of generic methods may be adapted for the purpose. Bilbao [61] points to a group of methods referred to as **spectral** [560] or **pseudospectral** [192], as particularly interesting, and capable of being applied to model musical instruments. Using spectral methods, approximation to a PDE system at chosen location can be obtained on the basis of the entire spatial domain, and not on the limited number of

neighbouring locations, as was the case of FD. The solution is decomposed into a set of basis functions [61, 347, 349], thus the method may be considered a generalisation of modal synthesis. Depending on whether the error is minimised in a finite set of spatial locations, or over the entire domain, two variants of the method exist. The former is referred to as a collocation method, and the latter – a Galerkin method [349].

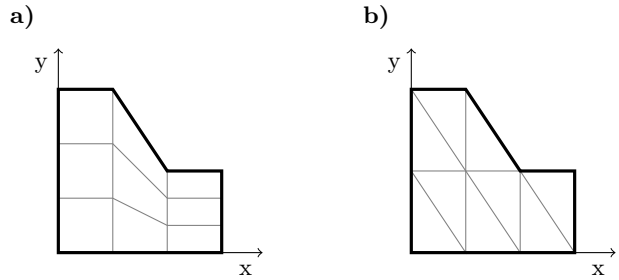


Figure 3.72. A quadrilateral (a) and triangular (b) division of a two-dimensional domain into elements in FEM
Source: author's elaboration, based on Kristiansen and Viggen [309]

Other than FDM, FEM, and spectral methods, attempts have been made to simulate certain instruments or their elements using the **boundary element method** (BEM) [80, 425]. However, many interesting cases involving brass instruments and flue pipes are non-linear and involve fluid mechanics, thus cannot be described well enough using aforementioned methods. Such cases may be approached using **lattice-Boltzmann method** [146] or **finite volume methods** [334]. Some attempts [3] make use of a **large eddy simulation** (LES) method to model turbulent flows [413].

4. Phrase Assembling Synthesis: a New Approach to Music Reproduction

4.1. Sound Synthesis in Music Reproduction

Musical applications of sound synthesis encompass two general scenarios. The first one is a real-time performance, and the second one – a reproduction of pre-entered musical score. In the latter a performer has to be substituted with some kind of mechanism or algorithm, which in majority of cases can be quite easily distinguished from live performance. Reproduction however, has an advantage of prior knowledge regarding music that is to be reproduced, which gives a hypothetical opportunity to carry out an analysis of some sort in order to improve the result beyond a regular playback of entered score.

Two major applications of synthesis in reproduction are computer-aided music arrangement, and composition. Both involve entering some form of a symbolic musical notation which is reproduced through a synthesizer either as a means to evaluate created piece, later to be performed by live musicians, or as a final performance. Particularly in arrangement tasks, a synthetic reproduction is often mixed with tracks containing recordings of live performances with real instruments. If the synthesizer part substitutes some acoustic instrument, it is vital that the imitation is as close to the real counterpart as possible [486]. Only then it can constitute a reliable basis for evaluation of composed work or mix well with recorded tracks.

Since early days of computer-aided music production, arrangers and composers have tried various synthesis methods. Even FM method, which is considered today a kind of archetype for synthetic sound, has been imitially developed with an imitation of acoustic instruments in mind [116]. Additive, subtractive, wavetable, as well as physical modelling methods, all of them had some advantages, however, over the years it was the **sampling** that has become the method of choice. It faithfully reproduced sounds of many musical instruments, though with some notable exceptions. At the same time it was efficient, and its low complexity allowed a simultaneous synthesis

of large ensembles. These two goals were hard to achieve simultaneously by other methods. Recently a new generation of waveform-based methods, the **concatenative synthesis**, is gaining attention due to its potential of addressing some shortcomings of traditional sampling, such as reproduction of natural note transitions. However, it is relatively new and not well tested, therefore only a small number of commercial implementations is available so far.

4.1.1. Shortcomings of Sample-Based Methods

In many cases properly applied sample-based methods are able to produce output that is extremely hard to distinguish from recordings of live performances with real instruments. However some instruments, articulations, or performance styles are still difficult to reproduce. In such situations the use of sound samples instead of continuous recording of real instrument is clearly audible. Therefore, there is a place for improvements.

Problems of Early Samplers

In early digital samplers the main concern was the memory capacity [470], which lead to the issues regarding:

- quality of samples,
- contents of samples,
- and processing of samples.

Unlike earlier analogue tape samplers, digital instruments such as Fairlight CMI Series I, with memory measured in kilobytes [470, 485], usually allowed to store only a short recording that required processing to control its pitch and duration. In order to produce required pitches the sample was transposed on demand, which was affecting timbre: no formants were preserved, no instrument registers were reproduced, and resampling-related distortions were occurring. Playback of longer notes required looping, which caused audible, repeatable distortions as well. As of controlling timbre, it was theoretically possible to sample various instrument articulations and dynamics levels, though only later samplers allowed more convenient switching between samples, e.g. through velocity mapping. All these issues have been solved as soon as larger memory became available and affordable.

Problems of Contemporary Samplers

The limiting factor shifted from a simple problem of memory capacity to more fuzzy issues, such as:

- content quality,
- coherence within sample sets,
- management of samples,
- control over fine-details,
- and reproduction of note transitions.

Sample Contents and Coherence

The first two issues depend on the process of sample preparation. With no memory limitations samples can be recorded, processed and stored using virtually any sampling frequency and bit depth – storage limitations have no impact on their quality. What remains of quality affecting factors is the effect of the recording process, quality of recorded instruments, and skill of musicians performing during the recordings. Therefore it is no more an issue of the synthesis method itself, but rather a matter of cost and time available.

Virtually unlimited memory resources allow to diversify contents of samples. Not only pitch multisampling is utilised, but it is common to sample instruments at different dynamics levels, performing various articulations. To add more realism, accidental sounds – produced by instruments under certain conditions – are also recorded to be mixed with main samples. As an effect, even a single instrument can be represented by an extremely large set of samples, with numbers reaching several thousands. These samples have to be coherent in order to blend together well into a performance, without oddly sounding 'outliers'. Again, it is not a technical issue, but a matter of controlling recording conditions and processing of recordings.

Sample Management

Not only preparing large sets of samples requires considerable amount of work, but using such sets may also raise considerable problems. Sample management becomes an issue when simple associations like key-pitch, or dynamics-velocity are not enough to select a sample, and it is not apparent how to map particular samples to given control events. One example may be a choice of a particular variant of *staccato* articulation if the score contains only general 'dot' marks.

The issue may be resolved either by arduous hand-picking of samples for every single note, which is not an uncommon practice, or by scripting the process, i.e. automating it by running a program written in a script language. The latter becomes more available in contemporary samplers, though programming scripts requires knowledge and skill from the user of synthesizer, while factory-provided scripts cover only selected scenarios. Thus with more samples to choose from, sampling synthesizers become less straightforward to use. Complex and sophisticated, they require more effort to obtain satisfactory effect, even though the threshold of what is considered satisfactory has considerably raised.

Fine Details Control

In the end, it was the advent of large, high-quality libraries of sound samples that made the last two issues particularly apparent. An inherent problem of the sampling method itself is caused by a discrete nature of sample sets. Playing back a sample leaves not much of the sound qualities to be controlled, so the most common way of changing a timbre is to switch to a different sample within a closed set. Such a limited control over the sound consequently limits the possibilities of a musical expression[1].

[1] In bowed string instruments and in wind instruments the sound is not only triggered, but also controlled in a continuous manner, though a sound sample of such instrument is only triggered.

Some implementations attempt to mitigate the problem by supplementing sampling engine with elements of other, more flexible synthesis methods [485]. The most common extension is an introduction of modifiers from subtractive synthesis – filters and envelopes. Some effects can be added with modulation-based methods, and additive resynthesis is being applied as a means to edit the sample from the ground, though it requires considerable effort to work in such manner with real recordings. There is however, a difference between modifying a square signal and a sound of cello. The former is plain, without inherent variability. All details are created by modifiers. In the latter the original source is complex and varies in time. Modifiers can introduce some coarse changes, but are not precise enough to entirely remove most of original features. Moreover, larger modifications applied to high quality samples are usually audible and cause the output to sound less realistic.

Note Transitions

The last issue is one of the most serious, with no immediate countermeasures. In its basic form sampling synthesis deals with separately recorded musical pitches. Each has its own amplitude envelope that begins with an attack phase. A regular procedure applied by a sampler to reproduce a melody, i.e. a sequence of pitches, is to play back a sequence of samples. In general, this method proves sufficient for instruments that produce various pitches by using separate vibrators. Such is the case of e.g. a marimba, harp, pan-flute, or – to a slightly less degree – a piano. In these instruments pitches are independent and there is no transition phase between them.

It is entirely different in case of bowed strings, or most of orchestra wind instruments. Here, pitch transitions are audible, and depend on performance-related factors. The issue is handled properly by physical modelling synthesis, but samplers do not cover this aspect. As a result it is very rare for a sampler to play a smooth, fluent, and natural strings or woodwinds phrase. More advanced samplers attempt to simulate the effect of pitch transitions by employing signal modifiers. This includes modification of amplitude envelopes and application of filters, like in subtractive synthesis [485], or spectral analysis and control over separate partials using additive resynthesis [470]. However, such operations may be audible, and the effect is not entirely natural, particularly when operations are performed on high quality samples.

4.1.2. Issues of Concatenative Method

Concatenative synthesis addresses several issues of sampling method. While in sampling sound samples have to be manually assigned to particular control events, in concatenative method management of recordings is the inherent part of the method – it is handled by a database system and dedicated algorithms applied in unit selection stage. Sampling is not able to modify fine details of sound samples in order to precisely shape the expression fitting the score reproduced. Concatenative synthesis does not have to shape these details. It simply searches for units best matching score context, and expressive details are preserved from the original recording [344]. Unlike sampling, which operates on separate pitches, concatenative method can utilise units

that include recorded natural note transitions, therefore it is able to produce fluent, smooth phrases.

However two issues that in sampling method are taken care of by proper preparation of samples, in concatenative method cannot be addressed with similar ease, namely:

- contents quality,
- and units coherence.

In addition, concatenative synthesis has its own set of issues, including:

- unit segmentation,
- complexity,
- reliability,
- computational cost,
- and artifacts caused by heavy signal processing.

Unit Contents and Coherence

There are several differences between a unit in concatenative synthesis and a sound sample used in digital samplers. The fundamental one is generality of definition. Samples are recordings triggered by selected control events, such as *note-on* message in MIDI, and in musical applications they represent single notes. Units are recordings that contain various levels of musical structures, from larger, like a sequence of pitches, to smaller, like the attack phase of a single note [344]. Various implementations of concatenative method utilise units of different levels. Units are not triggered, but concatenated to produce a continuous waveform.

Due to well-defined nature of sample sets for sampling synthesizers, they are planned, designed and produced purposefully. Contents of a set are known beforehand, and depend on the instrument sampled, assumptions regarding multisampling, as well as required performance techniques. Therefore a sound engineer is able to assure appropriate recording conditions and final coherence within a whole set.

Units for concatenation can be obtained in various ways, but it is common to cut them from previous recordings of whole musical performances. Such practice allows to retain expressive features included in recordings, but leaves no control over the recording process. Concatenated units may therefore sound slightly different due to varying recording conditions, even if their musical features, such as pitch or timbre, are well-matched.

Moreover, units do not have to form a complete set, i.e. not all pitches or transitions may be represented. If a structure or pitch is not represented in corpus, but still has to be synthesized, it needs to be transformed from a unit that is closest to the target [513, 505].

Automatic Unit Segmentation

Concatenative synthesis attempts to automate the procedure of obtaining units by automatic segmentation of music recordings. Depending on particular implementation various signal analysis tools and methods are applied. A common procedure is to align

a pitch-contour of the recording to the score using dynamic time-warping algorithm [532, 279, 20, 280, 398, 99, 13] or hidden Markov models [454]. Alternatively some external software may be utilised, and a number of ready to use tools is available in software toolkits such as MIR Toolbox [321].

However, all of mentioned solutions are prone to errors, except for the simplest musical structures. They may be applied to perform an initial segmentation, but in the end human verification and manual adjustments are required. And even combining human supervision with automatic segmentation may be insufficient to accurately locate boundaries between more fuzzy musical structures, e.g. the actual beginning of a note in the middle of a smooth pitch sequence played by a woodwind instrument. Such boundaries have to be approximated, which is a possible source of discontinuities in the output signal.

Complexity, Reliability, and Computational Cost

Concatenative synthesis is one of the most complex among sound synthesis methods due to multi-stage design involving the use of advanced algorithms and dependence on external tools. Units are obtained in complex automatic segmentation process [502]. Unit description data is stored in a database system. Selection of units involves database queries [501] as well as advanced cost-estimation [502] and optimal-path-search algorithms [596]. Finally, selected units are considerably transformed to match target features in time, frequency, and amplitude domain using signal processing algorithms such as filtering, resampling, additive resynthesis [336] or phase vocoder, and time granulation or pitch synchronous overlap add (PSOLA) [567].

As a consequence of method's high complexity a proper implementation of concatenative synthesis is difficult. It requires testing and balancing values for a considerable number of parameters controlling component algorithms and tools. Furthermore, part of these component elements are prone to errors. A significant source of such errors is the intricate nature of input data, i.e. recordings of expressive musical performance or high-level semantic descriptors. Another error source are misaligned working parameters, such as costs in optimisation algorithms. In effect concatenative synthesis may be far less reliable than simpler, more straightforward methods, such as sampling.

Complexity and dependence on computationally-intensive algorithms has a serious impact on time and processing power required to produce an output signal. It may not seem vital in non-real-time applications, such as music reproduction from entered score. However, in case of large, multi-instrument pieces, such as symphonic music which may be of interest for composers, time required to produce output may be a discriminating factor for a synthesizer. If it is to be used as an evaluation aid, a user may want to hear a piece, then make some small changes, and hear it again. Synthesis therefore needs to be practically instant, otherwise long wait periods will be a distraction from work.

Signal Processing Artifacts

Depending on the size of corpus and diversity of units, it may be possible to match some targets by simple selection and concatenation of units, though in most

cases selected units will have to be transformed before. Transformation involves applying signal processing techniques which may introduce audible artifacts. Two of most probable sources of such problems are pitch-shifting and time-stretching. Various implementations of concatenative method utilise different algorithms to perform these tasks.

A simple pitch-shifting through resampling, such as the one applied in early digital samplers, is adequate only for relatively small shifts. The source of a problem here is shifting of the whole spectral structure, including formants, which normally shall remain in place. Shifts larger than a few semitones are usually perceived as unnatural. In particular cases even one semitone may be a problem, e.g. on the boundary of registers in instruments with very distinct register-related timbre changes. A possible solution is to apply analysis and resynthesis, either additive or subtractive, and shift partials selectively. This however, requires considerable knowledge regarding acoustics of synthesized instruments, and in larger sets of units may become excessively labour-intensive.

Note duration can be easily shortened, as long as it does not violate the attack phase. The opposite operation involves more elaborate techniques. Two most commonly applied are granulation of time and PSOLA, and both lead to good results as long as the note transformed is relatively steady. Problems arise if a lengthened segment includes audible variability in time-domain of the oscillatory character, such as *vibrato* or *tremolo*. Such phenomena usually have a narrow range of typical frequencies, e.g. for a *vibrato* it is 5–7 Hz [385]. Time stretching may slow it down outside of a normal range, which will be perceived as unnatural.

4.2. The Concept

The **phrase assembling synthesis** (PA) is an attempt to address selected issues of sampling and concatenative methods, and provide another option for automatic score-based reproduction of music. It is aimed primarily at symphonic performances and concentrates on synthesizing fluent, natural phrases performed by individual orchestral instruments belonging to woodwinds and brass groups [445].

4.2.1. Motivation

Unlike strings, wind instruments in symphony orchestra perform their parts individually. E.g. in a pair of oboes, each of them has a different score to play. It causes features such as note transitions, articulation, or expression-evoked irregularities to be much more pronounced than in group parts, such as strings, where these features are bleary. Due to important role of expressive features performances of individual wind instruments are difficult to reproduce in a realistic manner by sampling synthesizer.

The method of phrase assembling has been conceived as a result of study carried out in the AGH University of Science and Technology and in the Academy of Music in Kraków, concerning the perception of pitch intonation in symphony orchestra [439].

During the research a simulator was employed to imitate errors in reproduction of pitch by wind instruments in fragments of symphonic music. Parts of wind instruments were reproduced by sampler. Ability of listeners to identify characteristics of pitch detuning was significantly affected by the amount of sound features reproduced by sampler [486].

A conclusion, that research related to perception of music can clearly utilise more realistic synthesis, led to formulation of ideas and assumptions regarding improvements and expansions to standard sampling method [157]. These constituted a basis for a research project funded by the Polish National Science Center, entitled 'Achieving sound realism in sampling synthesis of sound of symphony orchestra wind group' (2012/05/B/HS2/03972), and carried out in the Academy of Music in Kraków.

4.2.2. Key Ideas

It is not a synthesizer alone that may introduce an artificial quality to music reproduction. There are two stages where a music performance can lose its natural qualities. First, when a real instrument is replaced with its synthetic counterpart, and second, when a live performer is replaced with an automaton, such as the sequencer. An automatic score reproduction seems to be the worst case scenario. Here both replacements take place, and an automaton plays the synthesizer. However, score reproduction scenario has an advantage that can be exploited. It is the availability of a score before the performance starts.

Experienced users of sampling synthesizers are able to greatly improve quality of the reproduction by hand-picking carefully selected samples from very large sample collections. Selection is based on the context of a particular note within a score, as well as on additional knowledge regarding performance technique of the reproduced instrument. A match is further improved by adjusting sample parameters exposed by the sampling engine. One of key ideas behind the phrase assembling method was to automate this arduous and time consuming process.

Phrase assembling synthesis shall analyse the score before attempting to reproduce it. Thus it would be able to select best matching samples and adjust reproduction parameters according to a musical context. The process would not be unlike what an experienced performer does when he prepares the piece, firstly analysing it, and only then starting to play.

The second key idea regarded reproduction of note transitions and small-scale irregularities caused by music expression. Sampling method does not handle these tasks well, but concatenative synthesis prompts a solution in the form of rejecting traditional single-pitch samples and using larger musical structures instead.

Alas, concatenative method has its own problems. In practical applications the most important are its complexity and reliability, which in turn are strengths of sampling. Therefore a decision was made to design a hybrid method combining selected features of both, sampling and concatenative synthesis (Fig. 4.1). Table 4.1 presents a collection of assumptions regarding how to address issues of both source methods while designing a new derived one.

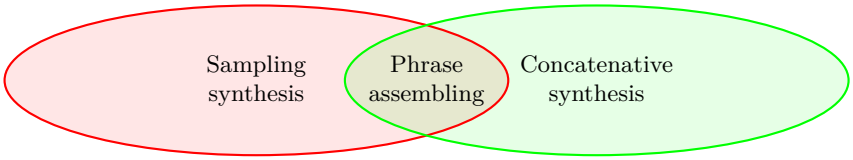


Figure 4.1. Phrase assembling synthesis in relation to sampling and concatenative synthesis methods

Table 4.1. Issues of sampling and concatenative methods attempted to be addressed by the phrase assembling method

Issue	Source method	Assumed solution
Content quality	Sampling and concatenative	Studio recording and processing of all the samples within a corpus
Sample or unit coherence	Sampling and concatenative	Purposefully recorded and processed samples instead of units cut from recorded performances
Management of samples	Sampling	Automatic selection of samples based on context of musical score
Fine-detail control	Sampling	Samples containing recordings of larger musical structures (motif-level) with inherent expression
Reproduction of note transitions	Sampling	Concatenation of multi-pitch samples on a sustain phase of a note
Automatic segmentation	Concatenative	Purposefully recorded and processed samples of well-defined content
Complexity	Concatenative	No database system, no external programs and tools, engine closer to sampling than concatenative synthesis
Reliability	Concatenative	Significantly simplified matching samples to target due to fixed and well-defined sample collection
Computational cost	Concatenative	Engine closer to sampling than concatenative synthesis, simplified matching samples to target, no external tools
Signal processing artifacts	Concatenative	Avoiding excessive signal processing due to large, purposefully designed sample collection

4.2.3. Method Outline

The core of the PA method is based on three integral elements. These are:

- a set of sound samples,
- a set of rules,
- and a set of algorithms.

The samples contain recordings of either individual musical pitches or short sequences of pitches, and form a corpus. The rules guide the following processes: score analysis, matching of samples to score, and concatenation of samples into a phrase. Finally, the algorithms analyse and process both, symbolic data and recordings, i.e. the musical score as well as sound samples, according to rules.

The PA synthesis process has four stages (Fig. 4.2). It begins with a symbolic musical score which is initially analysed to locate and separate consecutive musical phrases. Further processing is carried out on a phrase basis. In the second stage PA attempts to find a combination of samples from within its corpus that matches a sequence of notes within a phrase. Third stage involves processing of samples, i.e. cutting and concatenating them to form a reproduction of target phrase. In the final, fourth stage, reproductions of consecutive phrases are composed into a complete reproduction of the music represented by the score.

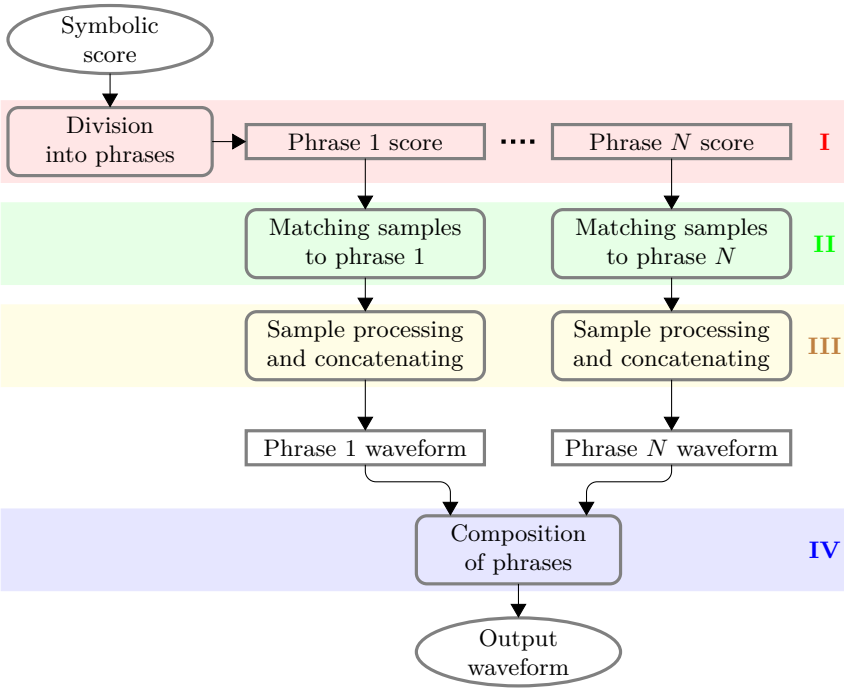


Figure 4.2. Four stages of the phrase assembling synthesis

4.2.4. Phrase

The name of phrase assembling method emphasises a prominent role of a phrase as a central point in the PA synthesis process. Depending on a particular style and historic period a phrase in music may be defined in a slightly different manner, although in general it is a musical unit that conveys a musical thought. Targeting wind instruments, PA refers to a more technical aspect of the term, according to which a phrase is considered a section of musical piece performed fluently, without break, i.e. 'on one breath'. Therefore a phrase is broken by events such as pauses, note repetitions, slur ends, or large jumps in melody [443].

From the perspective of sample-based method a phrase has an important feature: it has only one full attack transient, in the beginning of a first note. Subsequent notes are reached in a continuous manner, without attacks. Therefore in comparison to a single note performed separately, notes within a phrase have incomplete amplitude envelopes. They only have sustain phases present, and the remaining phases are either absent or reduced. Such effect is hard to reproduce using samples representing single notes only. Even after applying amplitude envelope to attenuate excess sections of note, connections between notes are still missing.

PA method addresses the issue by using multi-pitch samples. Such samples contain whole sequences of pitches with fluent transitions between. A phrase is analysed, matched with a fitting sequence of overlapping samples (Fig. 4.3), and assembled by:

- starting with a sample that has the same beginning as a target phrase to reproduce the first attack,
- and concatenating consecutive samples is such way, that the next one connects with the previous on a sustain phase of a note common for both of them.

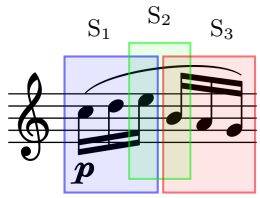


Figure 4.3. Matching multi-pitch samples, marked by colour rectangles, to a phrase; adjacent samples are overlapped on a pitch common for both of them

The most important aspect of the assembly method is the preservation of original, recorded note transitions, attained by allowing concatenations to be carried out only between transitions, on sustain phases.

Apart from the first sample which preserves its beginning, all subsequent samples in the assembled phrase have their attacks cut, as well as samples preceding them have their endings cut. If required, it is possible to further truncate a multi-pitch sample and remove a number of pitches from its beginning or end. Unlike attack, in case of most instruments release phase does not have very distinct features. Therefore it is not necessary to include the original final release from the sample in the phrase

and the phrase can end with a fade out applied to any sustain phase. Though in instruments where it is necessary, the final release can be concatenated in the same way as the initial attack. The assembly operation is depicted in Figure 4.4.

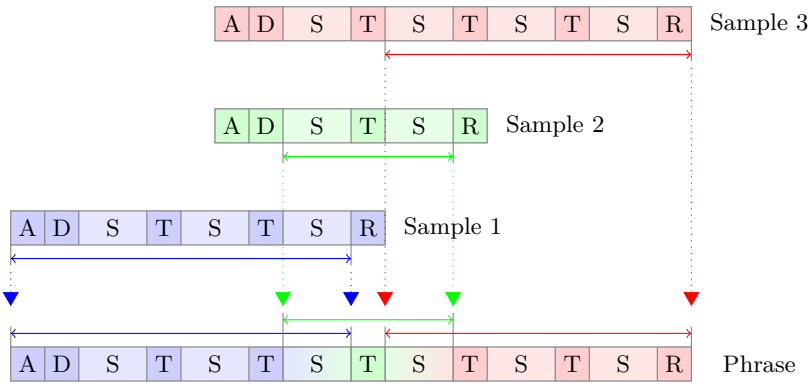


Figure 4.4. Assembling a phrase out of multi-pitch samples; lighter segments symbolise sustain phases (S), while darker segments symbolise transient phases, i.e. attack (A), decay (D), release (R) and pitch transition (T); each sustain phase represents a single pitch within a multi-pitch sample; samples are concatenated on common pitch using crossfade, symbolised by a color gradient, while recorded pitch transitions are preserved

Clearly, not all notes within a musical piece are tightly connected and require smooth transitions. There may be some separate notes as well as sequences to be performed using detached articulation, such as *detaché* or *staccato*. For cases like these, traditional sampling approach works fine, therefore for detached notes PA switches to conventional single-pitch samples.

4.2.5. Signal Processing of Samples

PA attempts to reduce amount of signal processing applied to sound samples during synthesis, in comparison to concatenative method. The most fundamental difference is a complete elimination of pitch-shifting. It is achieved in a manner similar to how conventional samplers apply pitch multisampling, i.e. by recording and storing every possible pitch, though with multi-pitch samples it requires considerably more effort.

PA utilises various types of multi-pitch samples, but it was assumed, that the corpus has to be designed in such manner, that there is at least one way of assembling any phrase playable for the instrument of choice. Therefore transition between any pair of pitches up to the interval of perfect octave is represented in at least one multi-pitch sample. Larger than octave jumps in melody are considered to break a phrase. As a result the corpus becomes very large, but the necessity of pitch-shifting is avoided.

Conventional music notation operates on discrete pitches[2], thus it is possible to store all pitch combinations. However, similar technique cannot be applied to note durations. Recording all duration combinations in various tempos is infeasible, therefore it is necessary for PA to implement some time-stretching method. It is also necessary to process amplitude of samples by applying envelopes. This however is one of the simplest operations, common for nearly all sound synthesis methods, and its performance impact is negligible.

4.2.6. Musical Expression

Music without expression is one of key indicators of synthetic reproductions, created using a sequencer controlling a synthesizer. Even though sequencers evolve and those more advanced are no longer simply reproducing notes in regular tempo, effect of their work is still easily distinguished from live performances.

PA method, like concatenative synthesis, is not only a synthesizer of sound. It synthesizes whole musical performances, thus combining synthesizer with elements of sequencer – it arranges samples in time. PA attempts to reduce the gap between the output of a synthesizer and the recording of a live musician by including expressive features in its reproductions. Technically, these features are minute variations in note durations, fine tuning of pitch, articulation, dynamics, and note connections. They are not a part of the musical score, but are the result of a human performer's way of understanding and interpreting a musical piece: its tension and climaxes, as well as its context and aims.

There are two different approaches to synthesizing musical reproductions with expressive features. Concatenative synthesis takes advantage of the expression present in the recordings used as a basis for units, and attempts to preserve it in the synthesized waveform. For other methods that operate on a pitch basis such approach is impossible to employ, and what remains is to synthesize the expression. Producing synthetic expression involves applying performance rules [199, 603, 84, 195] to modify parameters controlling both, synthesizer and sequencer. A very basic example of such rule may be the final *ritardando*, i.e. slowdown in the end of piece.

In PA both approaches are employed, although each one is applied on a different level of musical structures [158]. Multi-pitch samples contain structures comparable to short musical motifs. When performed by a live musician, such structures already demonstrate some expressive features, i.e. ascending sequences may slightly accelerate or become gradually louder, some intervals may be subtly detuned depending on surrounding pitches, articulation may emphasise larger interval jumps, etc. These expressive features are preserved in the output signal.

Phrase and higher level expression is simulated by PA method through a limited set of performance rules. These rules control primarily tempo and dynamics, and their envelopes in particular, shaping local and global climaxes. While low level expression is stored within sample recordings and cannot be modified differently than

[2]There are exceptions, such as *portamento*-like performance techniques, but they are rare in Classicism and Romanticism, which is the main scope of PA. If required, however, they can be produced by pitch-shifting the closest sample.

through switching to another sample, expression generated by performance rules can be controlled by user of the synthesizer.

4.3. The Design

4.3.1. Input and Output

The phrase assembling synthesis is designed as a non-real-time method of synthesizing music reproduction. It operates by taking a digital representation of a musical score along with a set of additional control data defining selected aspects of performance technique not included in score, and by transforming it into a waveform (Fig. 4.5).

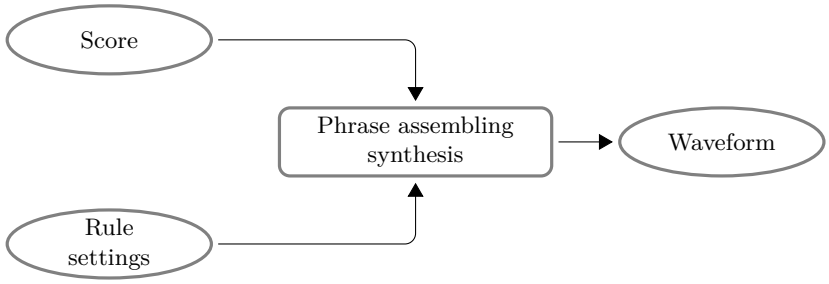


Figure 4.5. Input and output data type in phrase assembling synthesis

Division of input into two separate data sources allows to produce various reproductions out of a single score, simply by applying different rule settings. Through rule settings a user can customise synthetic reproduction to represent a particular performance style.

Details may depend on particular implementation, but it is assumed that the input score is provided by a user in the LilyPond data format [406]. LilyPond is a music engraving program published under the GNU General Public License [221], which is distinct due to its way of working with score, not unlike the LaTeX typesetting system [380]. A score is written in a human-readable text files and compiled into a chosen graphical format or a MIDI file. Source text files are not only human-readable, but the format it extremely convenient for further automatic processing of score, which is the case of PA method.

LilyPond format has been chosen over the MIDI representation, which is conventionally associated with synthesizers, due to its greatly enhanced capabilities to store fine details of musical notation, including various articulations, dynamics, time signature and tempo, as well as key information with enharmonic, which can be vital for producing expressive, realistic performance. It may be assumed that the format can handle virtually any kind of information and markings that could be encountered in

a musical score, apart maybe from some extremely vanguard graphics scores or other customised modern notation. Such notation is not within a scope of PA though.

LilyPond represents notes with letters, octaves with apostrophes or commas, and durations with numbers. Source code for a C-major scale is written in Listing 4.1.

Listing 4.1. A simple LilyPond code producing C-major scale

```
c'4 d' e' f' | g' a' b' c'' \bar "||"
```

A more sophisticated example in Listing 4.2 contains articulation markings, dynamics, and accidentals. The result of its compilation is presented in Figure 4.6.

Listing 4.2. LilyPond code with articulation, dynamics, and accidentals

```
\clef treble
\key a~\major
\time 3/4
   e'8[( \< \p fis'] gis'[ a'] b'[ cis'']) \! |
   d''4. cis''8( \> b'[ ais']) \! |
   cis''4( b'2) |
   d''4.-> cis''8( b'[ ais']) |
   cis''4. \< b'8( cis''[ d'']) |
   fis''4. e''8( fis''[ gis'']) \! |
   a''[( \mf gis''] fis''[ e'']) dis''[-. cis'']-. |
   dis''[( \> cis''] b'[ a']) gis'[-. fis']-. \bar "|." |
   cis''8[( b'] a'[ gis']) fis'[-. e']-. \! \bar "|."
```

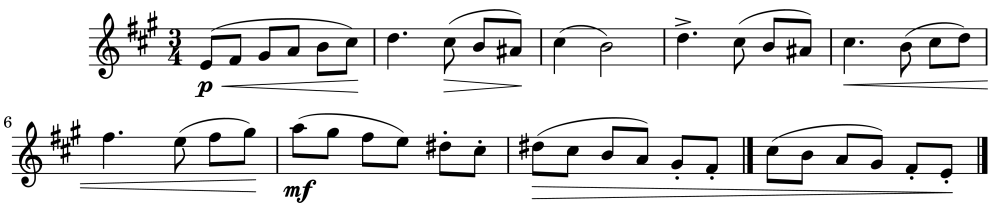


Figure 4.6. Example of a score generated by the LilyPond music engraving program

4.3.2. Samples and Descriptions

Unlike a conventional sampling synthesizer which may be considered an universal sample replay tool and shall work with various sample collections, samples in PA constitute an inherent part of the synthesis system. Processing algorithms assume, and depend on particular sample contents. The content of a sample is described and stored alongside a sample itself [445]. Description, available for each sample, covers three areas: instrument, sequence of notes, and concatenation data (Fig. 4.7). Details regarding description data is provided in Table 4.2.

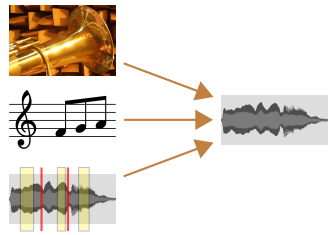


Figure 4.7. Types of information stored within a description of a sound sample

Table 4.2. Information contained within a description of a sound sample

Type	Information
Instrument	Name
Musical contents	Type of melodic figure Starting pitch Melody direction Dynamics Articulation Tempo
Segmentation data	Pitch boundaries Sustain regions

Information regarding musical contents is essential for searching of samples matching a phrase. There is a limited number of different melodic figures within multi-pitch samples. Thus instead of describing samples using names of component notes, figure identifiers are used instead, allowing to divide samples into groups. The same figure within a group of samples can be played starting from a different pitch, and in two directions, i.e. starting from each end of a sequence. Dynamics, articulation, and tempo also have to be considered, so that a timbre aspect of a sample is matched as well, and required duration correction is minimal.

Segmentation data is required for cutting samples and assembling them into phrases. It includes a set of locations within a sound sample, each one represented by index of a signal sample. With these the synthesizer is able to determine where to cut a sample if only part of recorded pitch sequence is needed, and where is the area safe to apply crossfade between adjacent samples.

Information regarding the instrument and musical contents is encoded into a file path and file name of the sound sample itself. Segmentation data is stored in a separate file with the same name as the sample, but different extension. The reason for such distribution of data is the efficiency concern. In concatenative synthesis units have to be managed with a database system. In PA samples and descriptions form a tree structure. During synthesis, when a sample with particular content is required,

the content only needs to be expressed as a text and formulated in a particular order, which immediately produces a path and file name of appropriate sound sample.

4.3.3. The Principle of Operation

Operation of the PA synthesizer is carried out in four stages, as shown in a simplified diagram in Figure 4.2. PA transforms a digital score into a waveform, therefore it processes two kinds of data. Part of its operations involves symbolic processing of score. The remaining part deals with signal processing of samples and final waveform [445]. Figure 4.8 presents a diagram illustrating essential operations performed by the synthesizer.

Stage I

Initial part involves entirely symbolic operations. The score is processed to determine locations of phrases. This part of analysis is based on a set of phrasing rules defining events that break the phrase [443]. Unless break condition emerges, consecutive notes read from the score are appended to the current phrase. Some phrases may contain a single note, and in such cases synthesis is straightforward, since no concatenation is required. Further operations, up to the point of composing phrase waveforms into a waveform representing complete output signal, are carried out over each phrase independently.

Stage II

During second stage a phrase is compared to the contents of corpus. Description of a sample allows to decide whether it matches particular fragment. In most cases more than one matching sample is available due to considering not only whole multipitch samples but also their parts. Therefore, although there is always at least one sequence of samples matching a phrase, it is also possible that more such sequences are available. In such case the algorithm chooses the best one, e.g. the one requiring the least concatenations [443]. A sequence of samples with cut positions specified is the information carried to the next stage.

Stage III

In this stage symbolic data from score and sample descriptions is used to control operations involving actual signal data within samples during phrase assembling. Samples are cut, their duration is modified if required, their amplitude around concatenation area is levelled, and they are positioned in appropriate locations in phrase waveform. Areas of concatenation are potential sources of discontinuities and have to be concealed. For this reason a cross-fade between adjacent samples is applied.

Simulation of musical expression involves introducing particular irregularities into note durations and positions, as well as signal amplitude. It may also require to select specific sample variants. These operations, controlled by performance rules, are applied during phrase assembling.

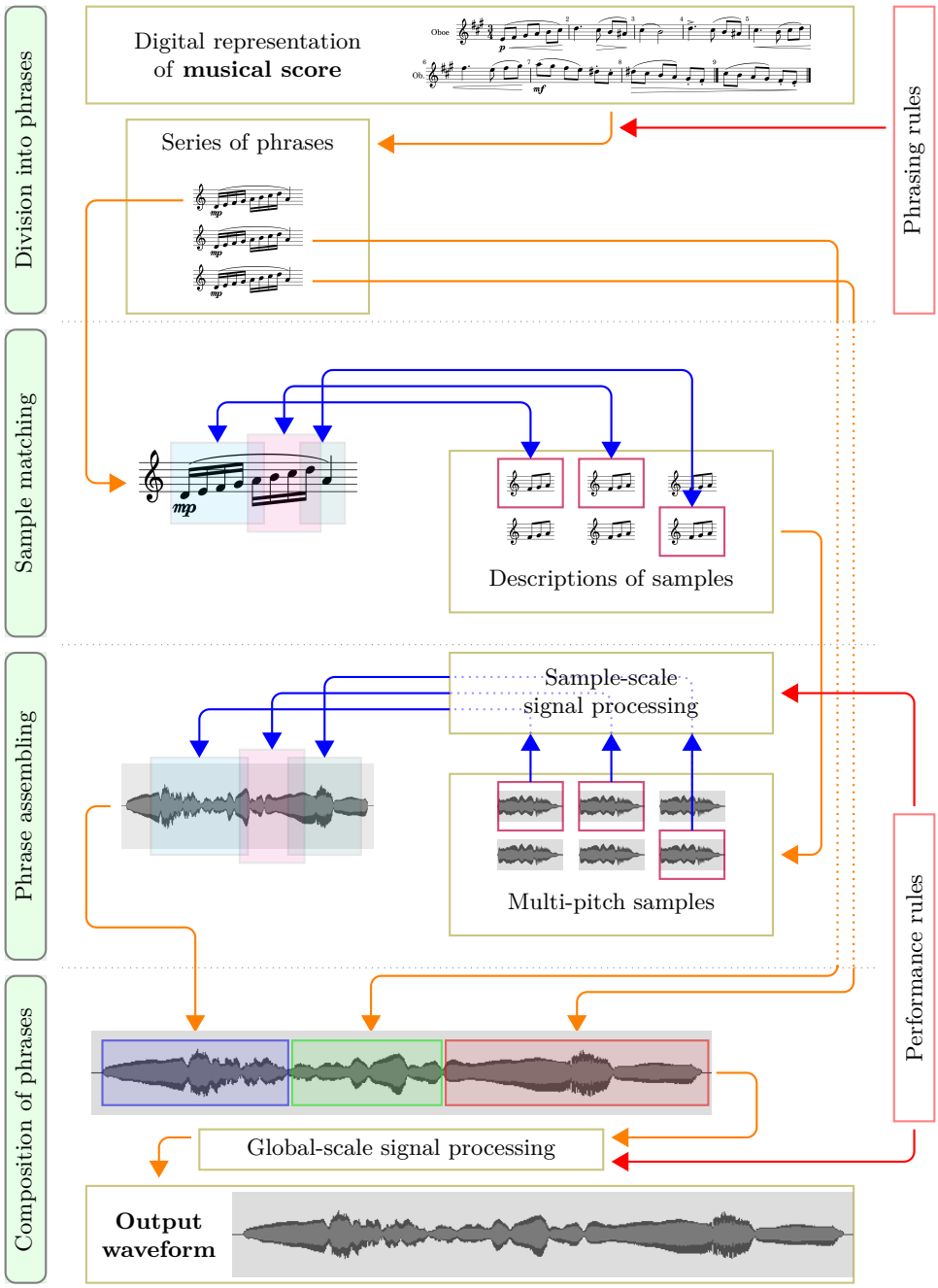


Figure 4.8. The principle of operation of phrase assembling synthesis

Stage IV

The objective of the last stage is to compose separate waveforms containing all assembled phrases into a single waveform – a complete reproduction of the synthesized piece. Due to the fact, that by definition the only smooth connections are those within phrases, and not those between them, no special concatenation of phrases is required. Consecutive phrases are simply appended in the order specified by the score with required gaps between. It is also here that performance rules are applied to shape large-scale expressive features, such as global amplitude envelope or breaks between phrases.

4.4. The Corpus

One of distinctive features of phrase assembling method is its corpus. It is a closed and strictly defined collection of sound samples representing chosen wind instruments. Contrary to open formula of corpora utilised in concatenative synthesis, closed and defined set of samples in PA allows to greatly simplify their management and usage. It does not require database systems, and due to known contents analysis and processing of samples may be significantly limited, e.g. each sample has a well defined sequence of pitches.

While concatenative method acquires units by segmentation of longer, continuous recordings, not necessarily recorded for such purpose, samples for the corpus of PA are prepared purposefully, with sequence for each sample performed and recorded separately, which makes it similar to sampling. The fundamental difference in comparison to corpora used in sampling method is the use of multi-pitch samples as basic melody building blocks.

4.4.1. Instruments

The corpus consists of samples representing ten wind instruments. Due to aiming at reproduction of orchestral parts only, samples have been recorded for pitches used in orchestral performances. Thus some extremely low or high pitches have been excluded. The list of instruments with recorded pitch ranges has been specified in Table 4.3.

Future implementations of PA method may extend the list of instruments. However not all of such extensions would make sense. The method is particularly well suited for reproducing individual instruments that play fluent melodic phrases. In case of nearly all percussion instruments, as well as instruments with separate vibrating elements for every pitch, there are no benefits of natural note transitions, and PA would simply perform as a conventional sampler. There may be a merit in adding bowed strings, although not as a group, like they are used in symphony orchestra. In a group performance two important advantages of PA are lessened due to the averaging effect. Considering that performance has to vary among instruments within a group, each instrument performs slightly different note transitions, and somewhat

individually interprets musical expression. Moreover, a group of live musicians is never perfectly synchronised. In effect note transitions become bleary, and expressive features less distinct, with leads to a more uniform final effect.

Table 4.3. The list of instruments in the corpus of PA with recorded pitch ranges

Instrument	Recorded range
Piccolo flute	D5–C8
Flute	C4–C7
Oboe	B♭3–G6
English horn	E3–C6
Clarinet	D3–G6
Bassoon	B♭1–D♯5
French horn	F♯2–C6
Trumpet	E3–B♭5
Bass trombone	B♭1–B♭4
Tuba	D1–G♯4

4.4.2. Structure

The corpus of PA consists of the following three parts:

- sound samples, i.e. PCM files containing recordings,
- accompanying text files containing descriptions,
- and directories in a strictly defined tree structure.

Both files, a sample and a description, share a common name, though with different extension: 'wav' for the former, and 'dat' for the latter. All three parts, i.e. both files and their location in directory tree, contain some information, as described in Table 4.4.

Table 4.4. Distribution of information in the PA corpus

Element	Information
File location	Instrument, sample type, sample subtype
File name	First pitch, dynamics, tempo, variant
Description file	Segmentation data
PCM file	Recording

The structure of corpus if reflected in its directory tree (Fig. 4.9). The highest-level branch represents the instrument, the middle one – a type of sample, and the lowest-level branch – a subtype. The same substructure is repeated in a branch of each instrument. It consists of three types of samples: single-pitch, two-pitch

interval sequence, and tetrachord sequence. A particular type has the same number of subtypes in each instrument branch, although the number of subtypes varies between types.

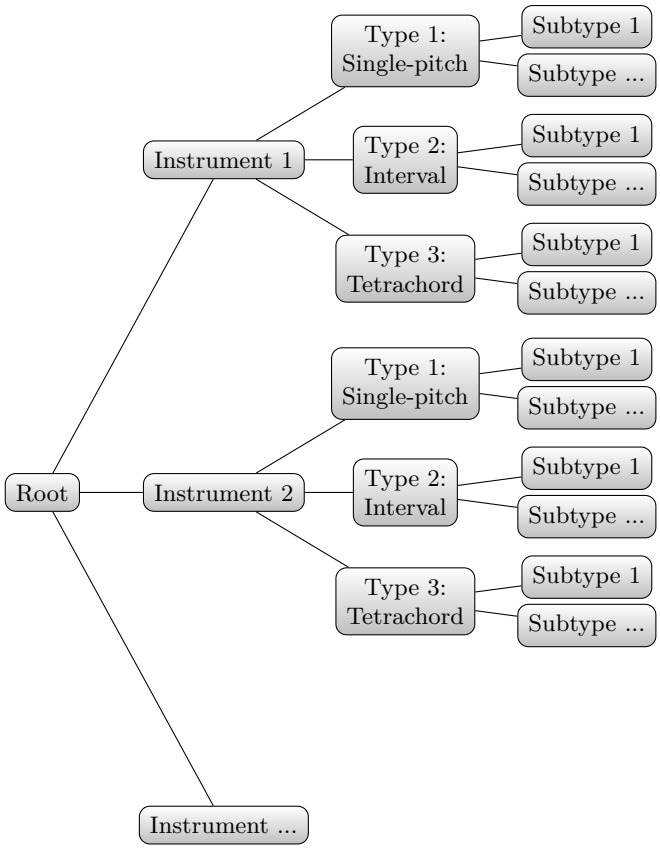


Figure 4.9. The structure of the PA corpus directory tree

A subtype-level directory is the actual location of sample files. Each subtype has a precisely defined interval structure, but samples within it represent different absolute pitches and performance features. Thus a given interval sequence is recorded starting from different pitches, and for each absolute pitch sequence there are sample variants with different dynamics and tempo. Moreover, the most frequently used samples are recorded in several variants with the same set of features, which is also indicated in the file name. They can be used interchangeably to avoid repeatability characteristic for digital samplers.

4.4.3. Contents

PA has been designed with the objective of reproducing specific musical structures. It aims at parts of wind instruments of symphony orchestra in compositions from the periods of Classicism and Romanticism, as well as other music composed in similar convention. A time frame given is related to the evolution of instrument design, composition, and music theory. Periods prior to Classicism utilised different ensembles and different tuning systems, while more recent approach to music composition, with notable examples of punctualism [77, 381, 220] and sonorism [115, 172, 608], often rejects conventional articulation and melodic structures entirely.

On the basis of assumption regarding musical structures PA aims to reproduce, with prevalent role of scale and chord sequences separated by larger jumps in melody, it has been concluded [441] that three types of units are sufficient to reproduce any such melody, without making a corpus unnecessarily large. The respective types of units are: a single pitch, a pair of pitches forming a melodic interval, and a sequence of pitches forming a tetrachord.

4.4.3.1. Units

Single-Pitch Units

Units consisting of a single pitch represent various performance techniques, articulations, and dynamic levels, as well as long notes. This kind of samples has two primary roles:

- detached articulations, such as variants of *staccato* or accents, are used in notes outside of phrases,
- long *vibrato* and *non-vibrato* notes are utilised to extend duration of shorter notes in sequences.

Moreover, special techniques such as ornaments may be introduced where the score requires them. A full list of single-pitch units is presented in Table 4.5.

Interval Units

Interval samples are the recordings containing a sequence of two different pitches, by definition forming a melodic interval. They are the fundamental and most universal melody building blocks among all PA samples. Any melody can be reduced to a series of intervals when no matching samples among these containing longer sequences can be found.

Size of intervals in PA is limited to perfect octave due to the fact that larger jumps are very rare inside continuous phrases. Each sequence is performed as a *legato* pair of eighth notes in a given tempo (Tab. 4.6). The most often occurring intervals are the minor and major second [139]. Due to this reason samples with intervals of both seconds are recorded in several variants for each combination of dynamics, tempo, and direction. They can be used interchangeably in order to make sample repeatability less audible.

Table 4.5. A full list of single-pitch units [443, 441]; if tempo is not marked, a note is the shortest or the longest possible, depending on unit; if dynamics is not marked, it depends on the performance technique represented by a particular unit

Subtype	Dynamics	Tempo	Comment
Short *staccato*	*mp*, *f*	—	—
Short *staccato sforzato*	*mp*, *f*	—	—
Long *sforzato*	—	—	—
Long *non-vibrato*	*mp*, *f*	—	Also for extending note duration
Long *vibrato*	*mp*, *f*	—	Also for extending note duration
Double *staccato*	*mp*, *f*	120 BPM	Four sixteenth notes
Full chromatic scale	—	fast	*Legato*, ascending and descending
Long *crescendo*	—	30 BPM	Whole note value
Long *diminuendo*	—	30 BPM	Whole note value
Semitone mordent	*mp*, *f*	—	Upper and lower
Whole tone mordent	*mp*, *f*	—	Upper and lower
Semitone *acciaccatura*	*mp*, *f*	—	Ascending and descending
Whole tone *acciaccatura*	*mp*, *f*	—	Ascending and descending
Semitone trill	*mp*, *f*	—	From main and from upper note (alternate)
Whole tone trill	*mp*, *f*	—	From main and from upper note (alternate)

Table 4.6. Interval units [443, 441]; all combinations of features are present in corpus

Feature	Value or description
Subtype	From minor second to perfect octave
Pitch distance in semitones	1–12
Dynamics	*mp*, *f*
Tempo [BPM]	60, 120
Direction	Ascending, descending

Tetrachord Units

Third type of units contains tetrachords. They are the longest sequences of pitches among samples used in PA. Tetrachord is a half of a musical scale, assuming the scale is a seven-step type with the first step repeated an octave higher – as the eight one. As such, tetrachord is a sequence of four pitches, each within an interval of second from the previous one. Some tetrachords have the same interval structure despite originating from different scales, and thus can be omitted.

In PA tetrachord units (Fig. 4.10, Tab. 4.7) are building blocks for longer scale-like sequences. Such units need to be able to be overlapped into longer progressions, so

that a scale crossing several octaves can be constructed. For that reason PA extends lower tetrachord by one pitch, which allows to overlap lower and higher tetrachords on a common pitch. Higher tetrachord overlaps to the lower one in the next octave due to the structure of seven-step scale itself (Fig. 4.11).

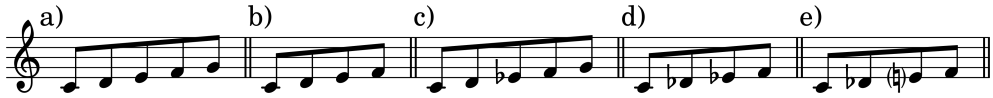


Figure 4.10. Tetrachord units used by PA with initial note C4: a) major lower; b) major upper; c) minor lower; d) natural minor upper; e) harmonic minor upper

Table 4.7. A list of tetrachord units [443, 441]

Subtype	Distances between pitches in semitones	Dynamics	Tempo [BPM]	Pitch order
Major lower	2–2–1–2	*mp*, *f*	60, 120	Straight, reversed
Major upper	2–2–1			
Minor lower	2–1–2–2			
Natural minor upper	1–2–2			
Harmonic minor upper	1–3–1			

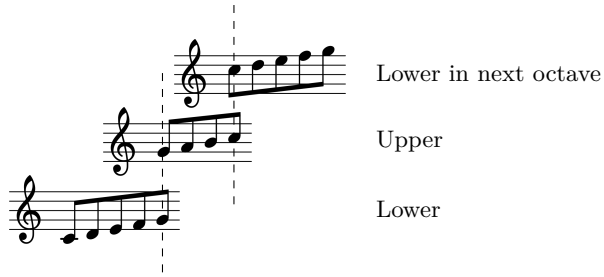


Figure 4.11. Overlapping of 5-pitch and 4-pitch tetrachord units applied to produce longer scale-like sequence

The reasoning behind introduction of tetrachord units in addition to intervals, which can be used to build scale progressions as well, is a distinct expressive character of longer ascending or descending sequences in interpretations of live musicians [84, 296], and frequent occurrence of such structures in melodies. This expressive trait would be lacking in reproduction combined out of single intervals.

4.4.3.2. Multisampling

Subdirectory representing a particular unit subtype stores a collection of samples that differ in the following features:

- starting pitch,
- dynamics (if applicable),
- tempo (if applicable),
- variant (for the most common samples).

A fundamental feature of the PA corpus considering sample processing applied during synthesis is the full pitch multisampling. PA synthesizer does not need to transpose pitch of samples used to assemble a phrase, because every possible combination of pitches is recorded with a corresponding natural pitch transition.

Each subtype is recorded throughout a full range of pitches used in orchestral parts of a particular instrument (Tab. 4.3). Unlike in multisampling of conventional single-pitch samples, in multi-pitch samples a number of samples starting with different pitches depends not only on the range of instrument, but on the ambitus of the musical figure represented by the unit in question, e.g. there are less samples of large intervals than of small ones, because the latter can start from higher pitches unavailable for the former. This however has no effect on reproduction capability of the PA, as long as all notes in the score reproduced are kept within the instrument range.

Tempo and dynamics in music are less discrete than the pitch, with fluent gradations possible, therefore in case of these features multisampling had to be limited to some arbitrarily chosen closed set of fixed values. In the first implementation of PA two values are used in both cases: slow and fast tempo, i.e. 60 and 120 BPM, as well as soft and loud dynamics, i.e. *mp* and *f*. In both cases impact on expression and timbre resulting from switching from one value to the other is clearly pronounced, and it would be difficult to convincingly simulate it with signal processing only. Milder changes, however, can be obtained in such manner, therefore no intermediate values have been recorded in order to maintain a reasonable size of corpus.

With current assumptions regarding multisampling as well as selection of pitch sequences for multi-pitch samples, each of ten instruments in corpus is represented with a collection of approximately 5000–6000 different sound samples.

4.4.4. Recordings

All of the sound samples for the PA corpus have been recorded specifically for this purpose. Recordings have been carried out in the recording facilities of the Academy of Music in Kraków as a part of the Polish National Science Center research project no. 2012/05/B/HS2/03972. All instrument parts have been performed by the staff of the Academy of Music.

For a prototype implementation of the PA method two stereo recording techniques have been utilised [256]. A pair of DPA 4006 microphones was used in the A/B set-up, and a pair of Schoeps 4V was used in the X-Y set-up. All relevant files are available in

the corpus for the user to choose from. Sampling frequency has been set to 88.2 kHz to leave an adequate headroom for signal processing.

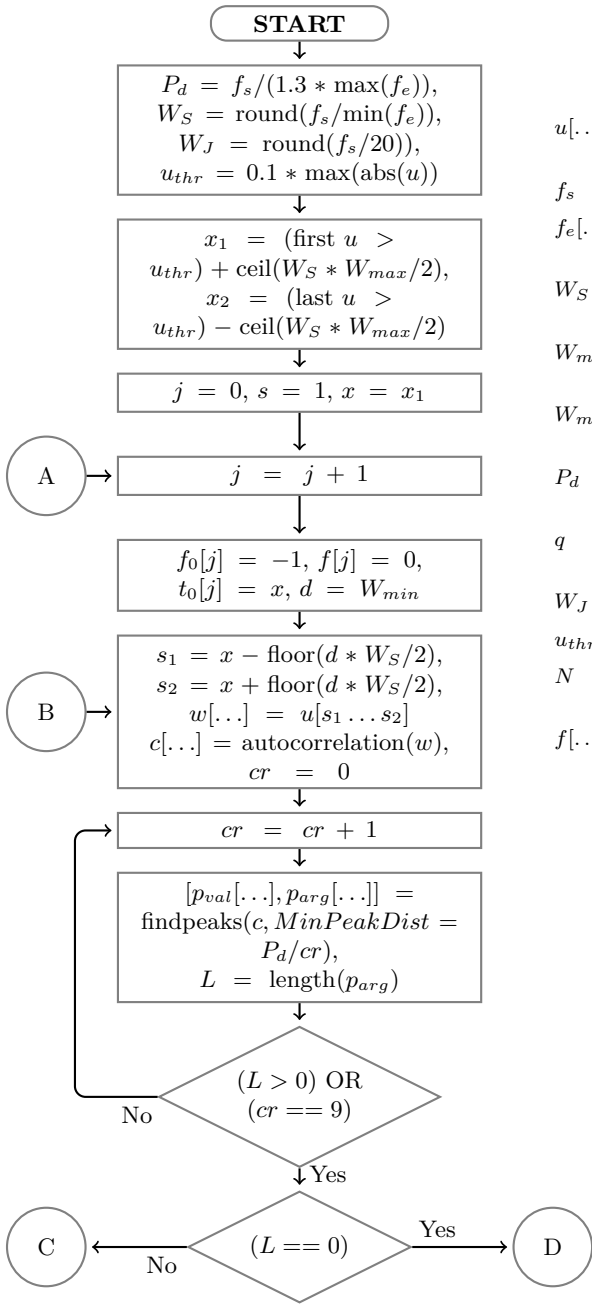
4.4.5. Analysis and Preparation of Samples

In comparison to concatenative method, PA requires a relatively modest amount of analysis and processing before a unit can be included in the corpus. Regarding information types mentioned in Table 4.2, all the instrument and music contents related data is fully known beforehand. The only information that remains to be determined is the third part – segmentation data, required for concatenation of samples. In particular, the data is a set of positions within a sound sample, expressed in signal sample indices, which – with known sampling frequency – translates to time positions. The set consists of boundaries between pitches within a sequence, and extent of sustain regions corresponding with particular notes. However, even though the amount of information needed to be extracted from recordings is significantly smaller than in concatenative synthesis, the amount of recordings to process might be much larger. Thus a fully manual approach to the task would be impractical, if not impossible, and application of some algorithms to automate the process is necessary.

The goal of the analysis is not to determine f_0 of the signal in question, but rather to establish consecutive areas of constant pitch, representing notes in the recorded sequence. An algorithm proposed by Pluta and Delekta [443] has been utilised in the task with satisfactory results. The algorithm, presented in Figures 4.12–4.15, applies an autocorrelation method of f_0 extraction [212] (Fig. 4.12), and supplements it with pitch tracking, which uses *a priori* information regarding a sequence of pitches expected in the analysed recording (Fig. 4.13–4.14). With known expected frequencies, the algorithm adjusts f_0 extraction parameters, including detection window and minimal distance between peaks in autocorrelation.

Initially the window is small, and minimal peak distance is assumed large, however if the process fails, both values are progressively loosened. Once detected, f_0 is compared to the current or next expected value. The algorithm attempts to take into account common octave errors. In the end, areas where f_0 has not been established, but which are surrounded by areas with the same determined value of f_0, are filled in (Fig. 4.15). The result of the process is a list of coordinates that confine non-overlapping areas of subsequent notes, that are separated by transition phases.

Even though the algorithm presented proved to be relatively robust considering a very large set of samples analysed, it still fails to provide correct results in some isolated cases, therefore requires human supervision. For some instruments the autocorrelation may be substituted with a different f_0 extraction method [178]. Alternatively, an entirely different tool may be applied, such as the *YIN* algorithm [153, 152] that estimates not only value of f_0, but also aperiodicity and loudness.



variables

- $u[\ldots]$: **input**, i.e. unit, sound sample
- f_s : sampling frequency
- $f_e[\ldots]$: sequence of expected frequencies
- W_S : period of the lowest pitch
- W_{min}: min. number of W_S in examined window
- W_{max}: max. number of W_S in examined window
- P_d : min. peak distance for autocorrelation
- q : quarter-tone frequency ratio ($2^{(1/24)}$)
- W_J : window shift
- u_{thr} : signal threshold value
- N : number of examined windows
- $f[\ldots]$: **output**, i.e. values of f_0 for consecutive windows

Figure 4.12. Pitch-regions finding algorithm, part 1 of 4: initialisation and autocorrelation

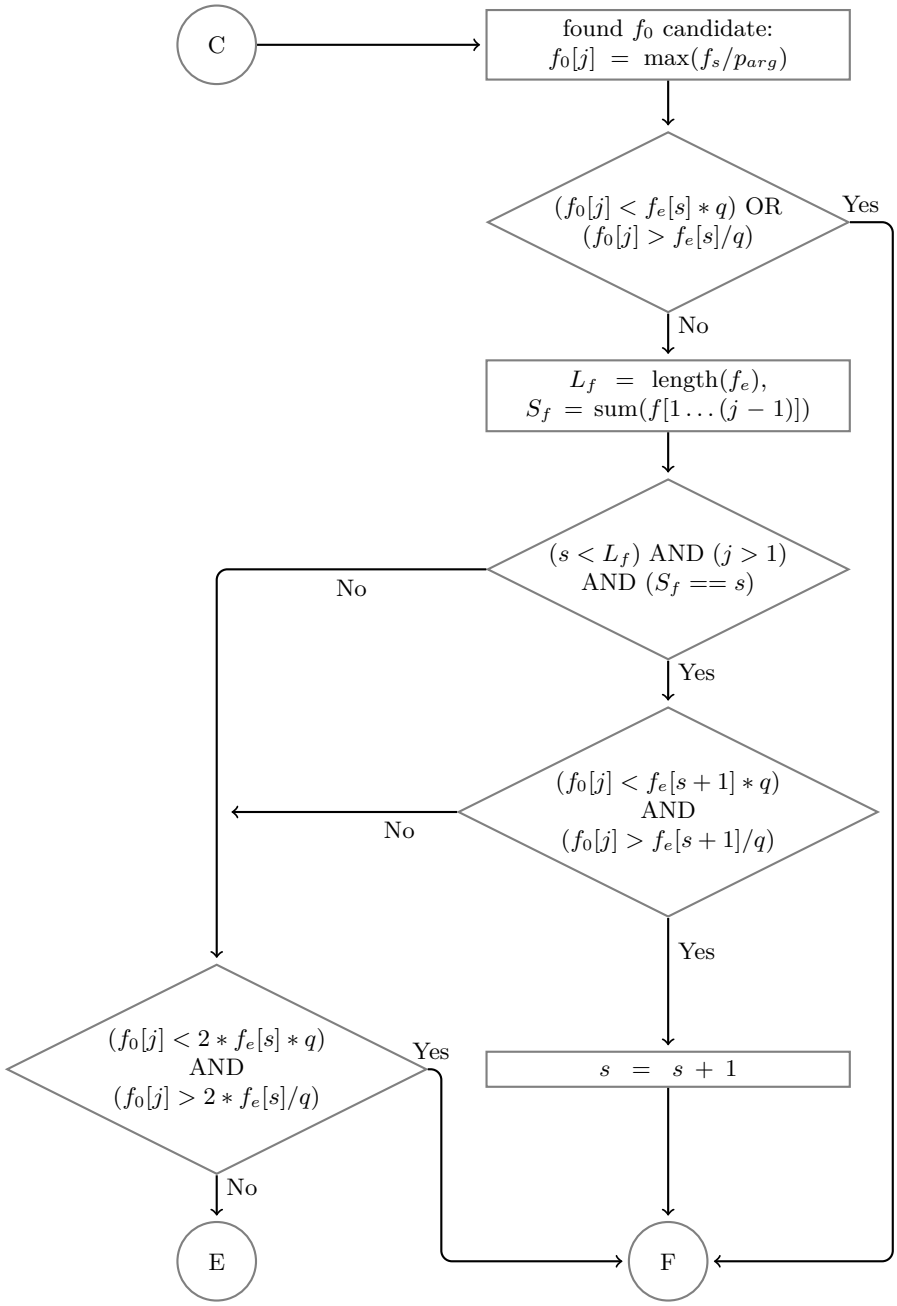


Figure 4.13. Pitch-regions finding algorithm, part 2 of 4: pitch tracking and octave checking

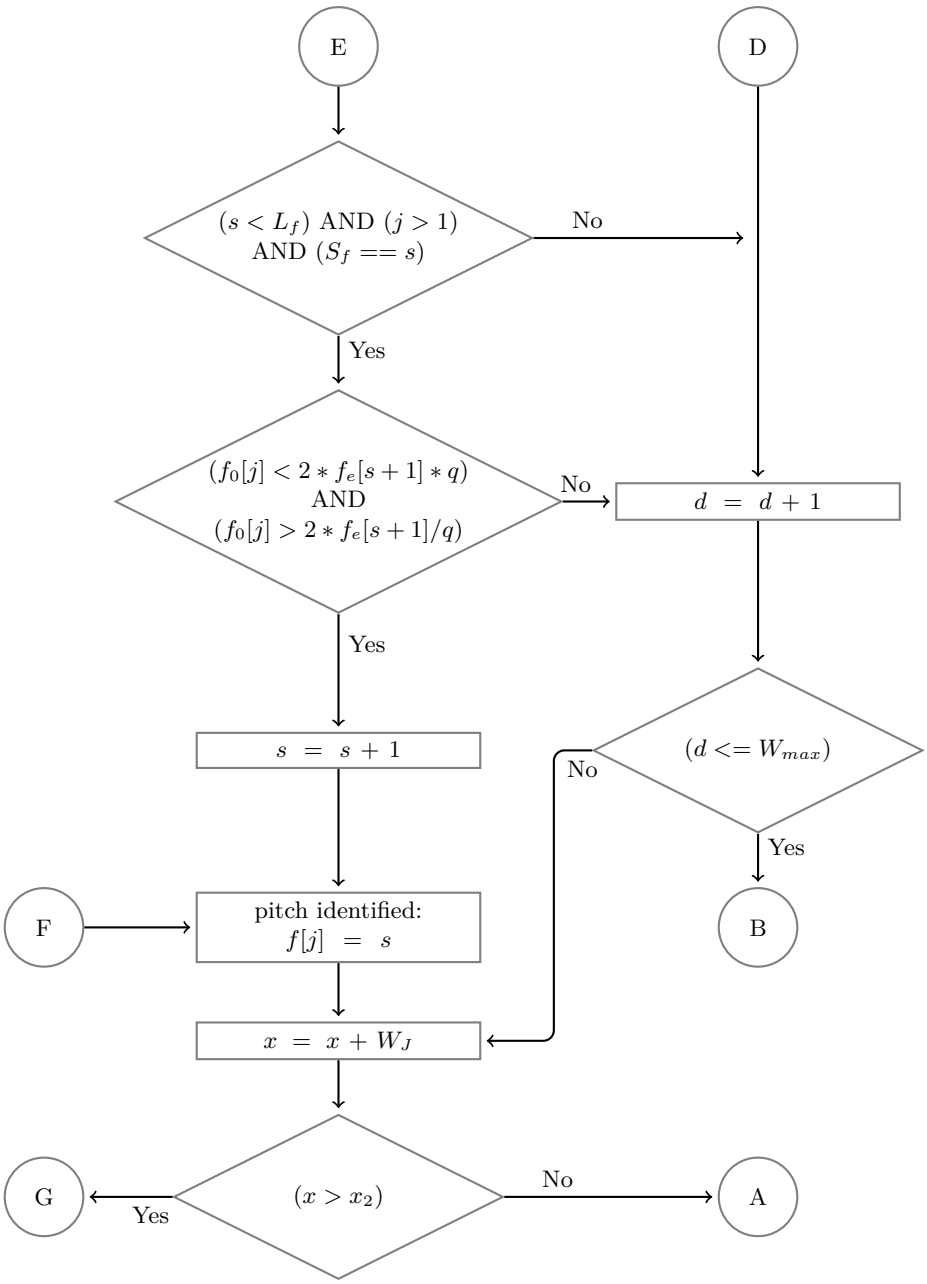


Figure 4.14. Pitch-regions finding algorithm, part 3 of 4: pitch tracking and octave checking

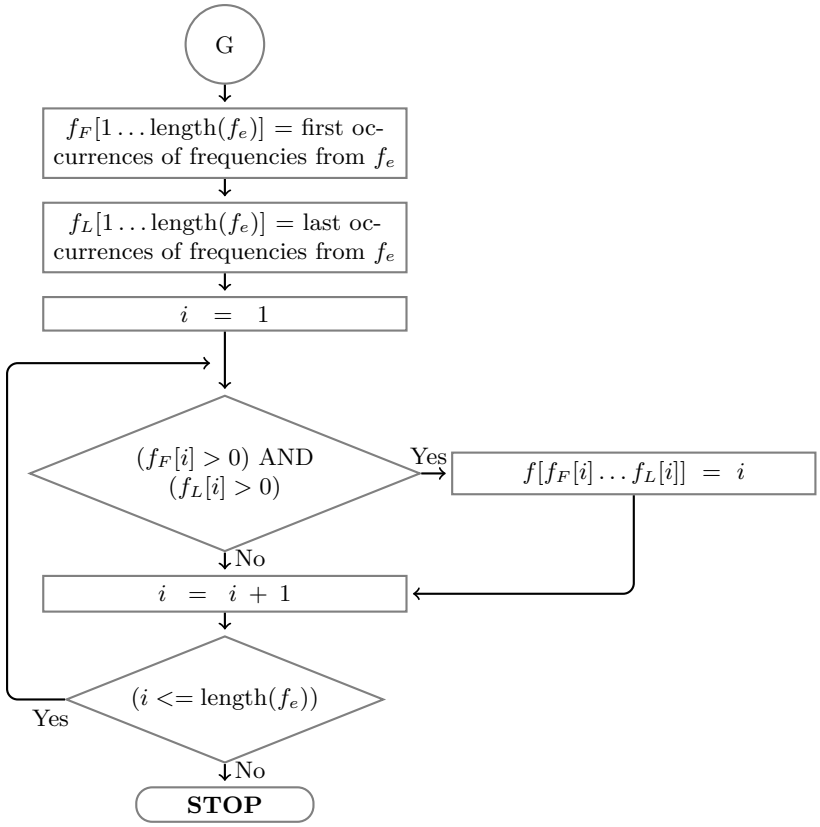


Figure 4.15. Pitch-regions finding algorithm, part 4 of 4: filling gaps

4.5. Applied Techniques

Transition from a musical score to a waveform involves several levels of processing. PA applies a number of algorithms and techniques to analyse a score and process signal on a sample, phrase, and a final waveform level. In the score domain specialised algorithms are applied to segment input into phrases and to select matching samples. Concatenation of samples makes use of a particular variant of crossfading technique adjusted to deal with highly correlated signals. Another technique applied on the sample level allows to control note durations. Even though multi-pitch samples contain some low-level irregularities in performance attributed to musical expression, higher-level expressive features have to be added on a phrase level. Therefore, while assembling samples into a phrase, a set of performance rules is applied. Performance rules can affect various parameters, but in PA two most important are envelopes applied to dynamics and tempo.

4.5.1. Musical Score Analysis

4.5.1.1. Score Segmentation Algorithm

The initial step of the PA synthesis is the segmentation of entire input score into a sequence of musical phrases. The process may be considered as a series of decisions regarding a type of connection between each consecutive pair of notes. If a particular condition is met, a phrase is considered to be broken. Conditions are identified by analysing a score and searching for events such as pauses, note repetitions, slur ends, or large jumps in melody [443].

A LilyPond's textual form of musical score (Listing 4.2) is well-suited for the purpose of such analysis. All of the events sought for either emerge as an effect of analysing immediate note neighbourhood, like in case of pause or repetition, or are simply represented as an information attached to a note, like end of slur or detached articulation.

A part of score segmentaiton algorithm is presented in Figure 4.16. A score is represented by an array with each element being either a note or a pause. Element value of 1 represents a note that begins a phrase. Value 0 represents a note that continues a phrase. Separated notes are treated as one-note phrases.

4.5.1.2. Phrase Matching Algorithm

Once a score has been segmented into a sequence of phrases, another score processing algorithm carries out a sample matching procedure (Fig. 4.17–4.19). During this stage samples are represented by figures, i.e. symbolic sequences of notes that describe musical contents of multi-pitch samples. One-note phrases are trivial and require no further analysis. For longer phrases the algorithm finds all sequences of figures that match a phrase, further referred to as solutions, and selects one of them.

Figures matched to a phrase do not have to be used as a whole. Partial figures, with some notes removed from either end, can be used as well (Fig. 4.4). Information regarding removed notes has to be attached to each figure within a sequence. Therefore an algorithm stores a single solution in three arrays. The first one contains identifiers of consecutive figures. The other two contain the first, and the last used note of a respective figure. All three arrays are stored within a single, two-dimensional array, where the second index allows to select type of information: a figure ID, its first note used, and its last note used.

The initial task of the algorithm is to search for all figures that begin with at least first two notes of the phrase, in order to reproduce the first note transition (Fig. 4.17). Analysed figure is checked to determine how many consecutive phrase notes does it match. For some short phrases even a single figure may match the whole phrase, which concludes the algorithm, with the solution complete.

If some phrase notes are still not matched, another figure is searched for. A search condition is that a figure includes the last matched note from a phrase, for overlapping samples, and the first not matched one, so that the solution expands. In this stage a solution starting with a given first figure can branch, i.e. have several alternative second figures. Within each branch a second figure is checked like the first one, to determine how many phrase notes does it match.

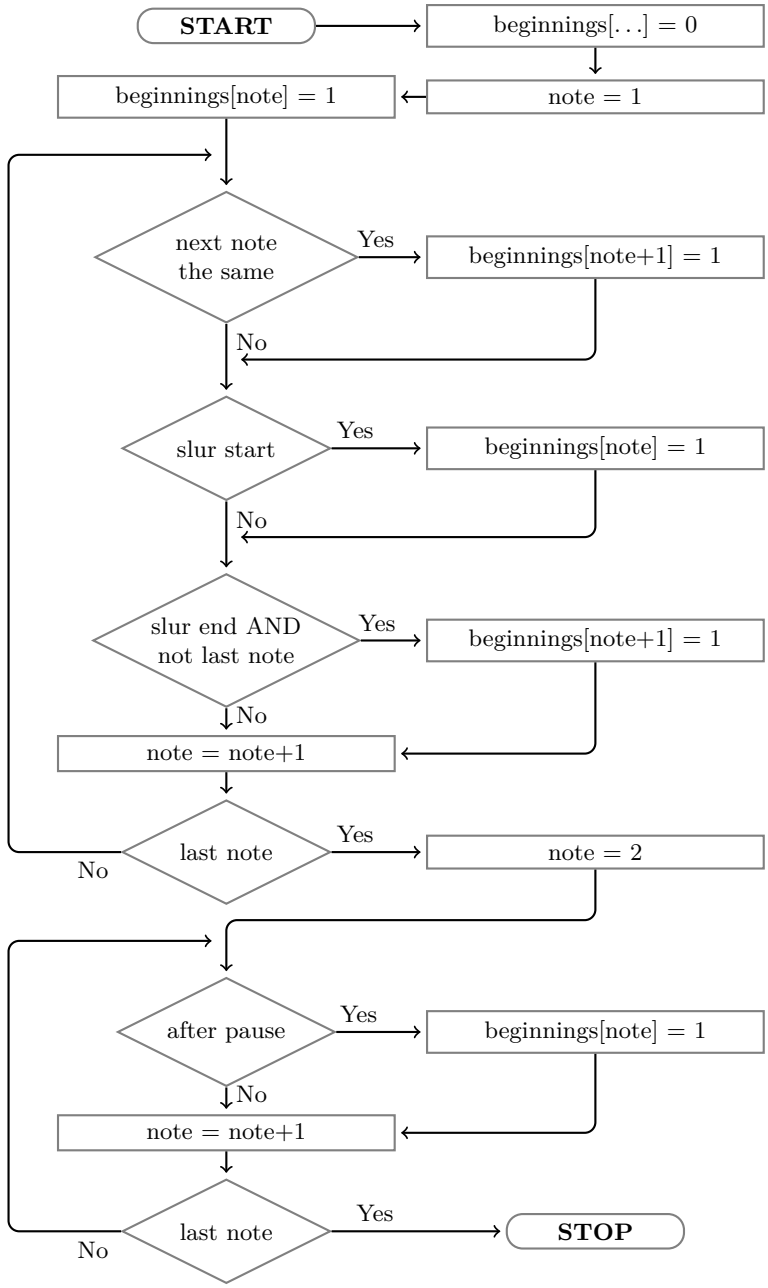
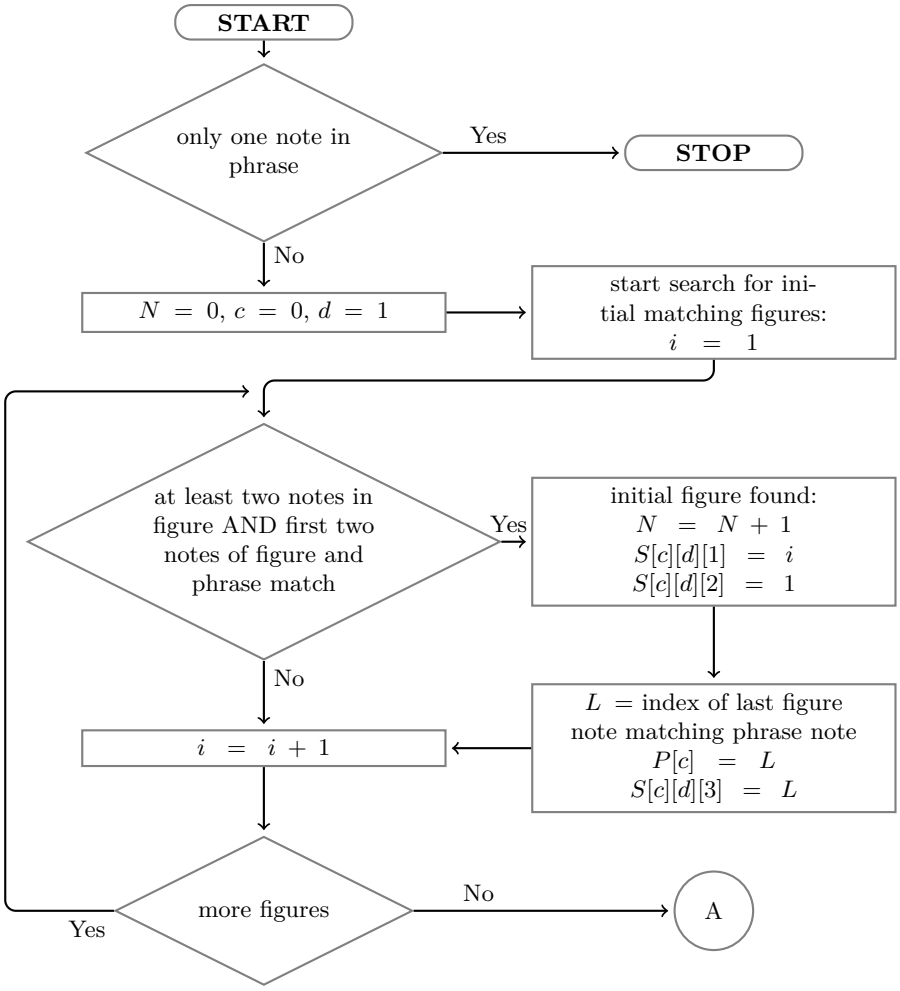


Figure 4.16. A part of score segmentation algorithm, with three types of phrase-break events recognised: note repeat, slur, and pause; array 'beginnings' represents all notes in the input score; its elements set to 1 represent beginnings of a phrase



variables

$S[\ldots][\ldots][\ldots]$	:	array of solutions; first index – solution ID, second index – depth (position of figure in matched sequence), third index – 1: figure ID, 2: first used note, 3: last used note
$P[\ldots]$	:	matching progress (last phrase note matched)
N	:	number of solutions found
b	:	currently processed branch
c	:	currently processed solution
d	:	depth

Figure 4.17. Algorithm matching a sequence of figures representing sound samples to a phrase, part 1 of 3: initial figure

If the solution is not complete yet (Fig. 4.18), matching and checking procedure repeats as many times as required in each branch. In the following iterations of the procedure solutions can branch further (Fig. 4.19). Finally, a set of candidate solutions is prepared. Next stage of the algorithm concerns selecting the best one among them.

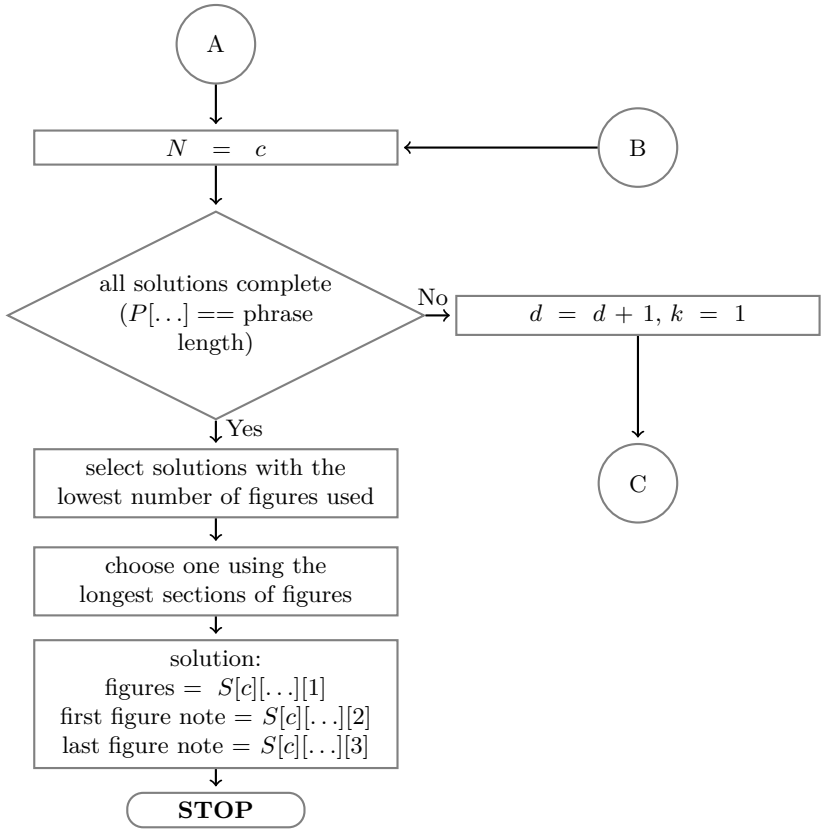


Figure 4.18. Algorithm matching a sequence of figures representing sound samples to a phrase, part 2 of 3: completing solutions

The best solution can be chosen according to various criteria. A simple assumption might be that the phrase shall be assembled using as few samples as possible. Such condition will lead to solutions with the lowest number of sample concatenations. Furthermore, among solutions with the same number of required concatenations the algorithm will prefer those that utilise the largest parts of its component samples (Fig. 4.18). This shall allow to retain the most of low-level expressive features from the recordings.

With such criteria the algorithm does not have to continue iterating to find all matching solutions. It can simply stop after the first iteration where complete so-

lutions emerge, which will make the process more efficient. Nevertheless, mentioned criteria are not the only viable. It is even possible to select a random solution from a complete set. Such approach might be reasonable in case of highly repetitive music, and will lead to slight variations in repeated phrases.

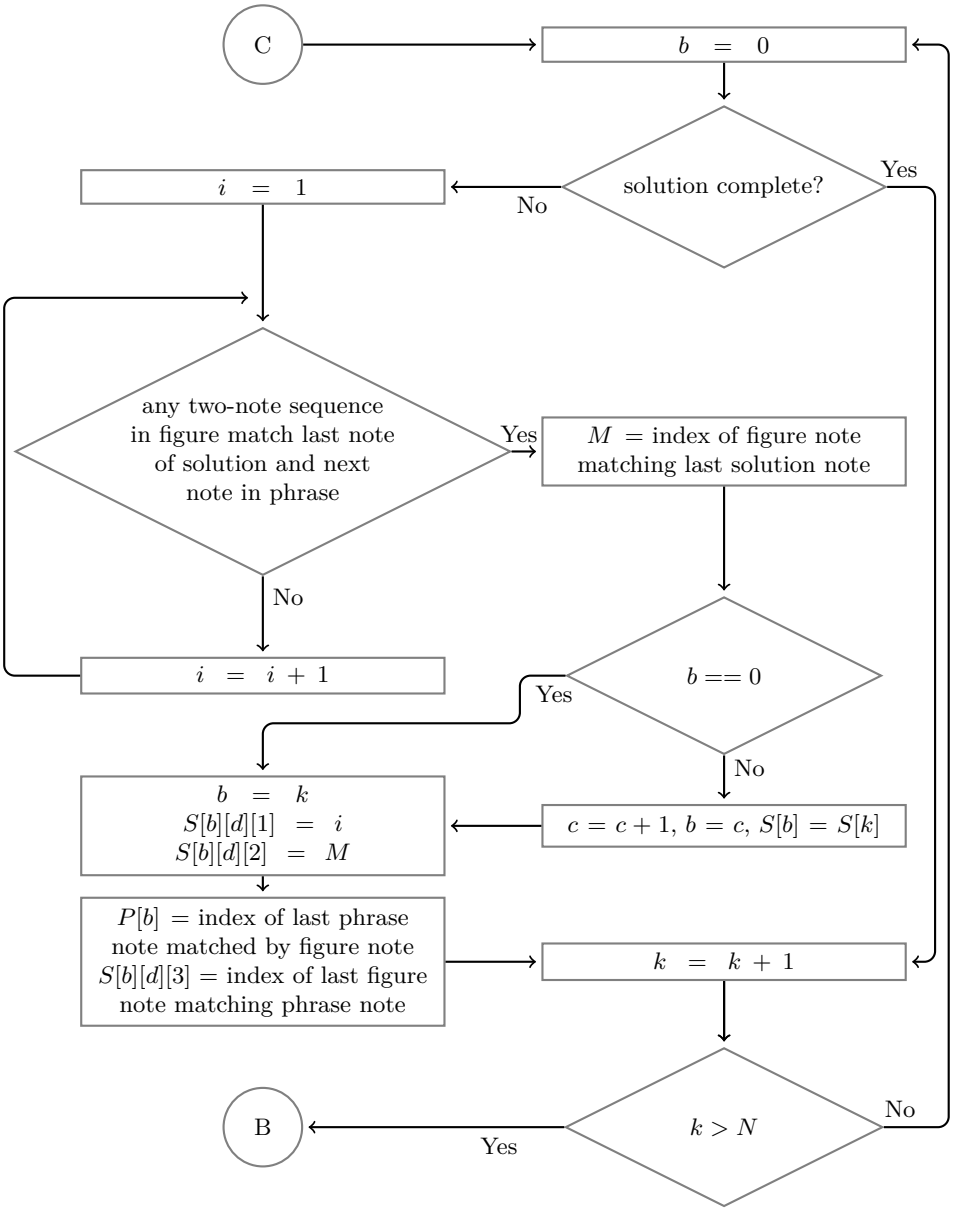


Figure 4.19. Algorithm matching a sequence of figures representing sound samples to a phrase, part 3 of 3: branching

4.5.2. Sound Samples Processing

In comparison to concatenative synthesis phrase assembling method requires far less signal processing applied to sound samples. As a consequence of complete representation of all required pitches and pitch transitions within the PA corpus, there is no need to transpose samples or their parts. There is still, however, an issue of concatenation of adjacent samples on a sustain phase of their common, overlapping pitch – area of connection should be inaudible.

4.5.2.1. Concatenation

Samples are concatenated using a crossfade, which allows one sample to gradually transform into another one. However, such simple approach cannot be applied directly due to a specific relation of connected signals. Both concatenated samples contain recordings of exactly the same instrument, obtained under the same conditions, and prepared in the same manner. Moreover, concatenation is performed on a sustain phase of the same pitch, hence both samples have almost identical fundamental frequency. As a consequence, crossfade regions in both samples have a very high correlation. A special attention has to be paid to relation of their phases, otherwise it is likely that some cancellation will occur, resulting in a drop of amplitude in the concatenation region.

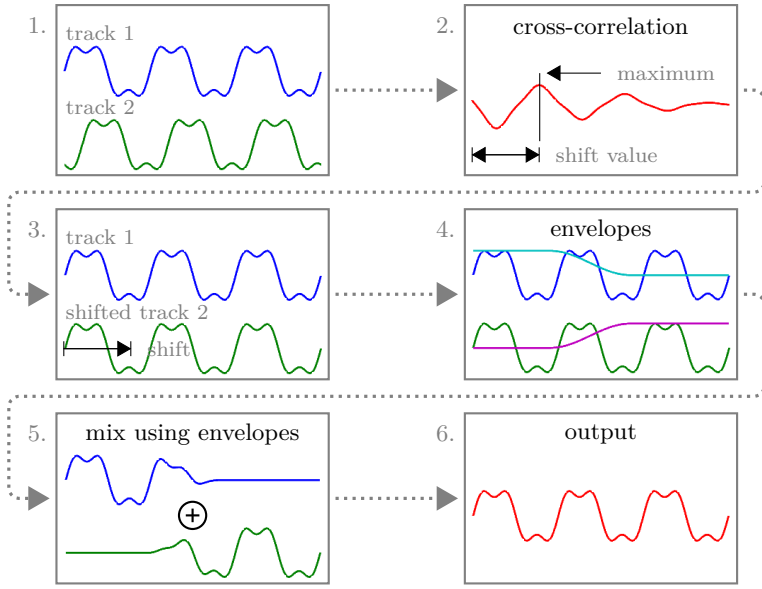


Figure 4.20. Phase-aligned crossfade

The issue can be addressed by applying phase alignment before the crossfade. Schematic diagram of the operation is presented in Figure 4.20, while details of implementation can be examined on the basis of the flowchart presented in Figure 4.21.

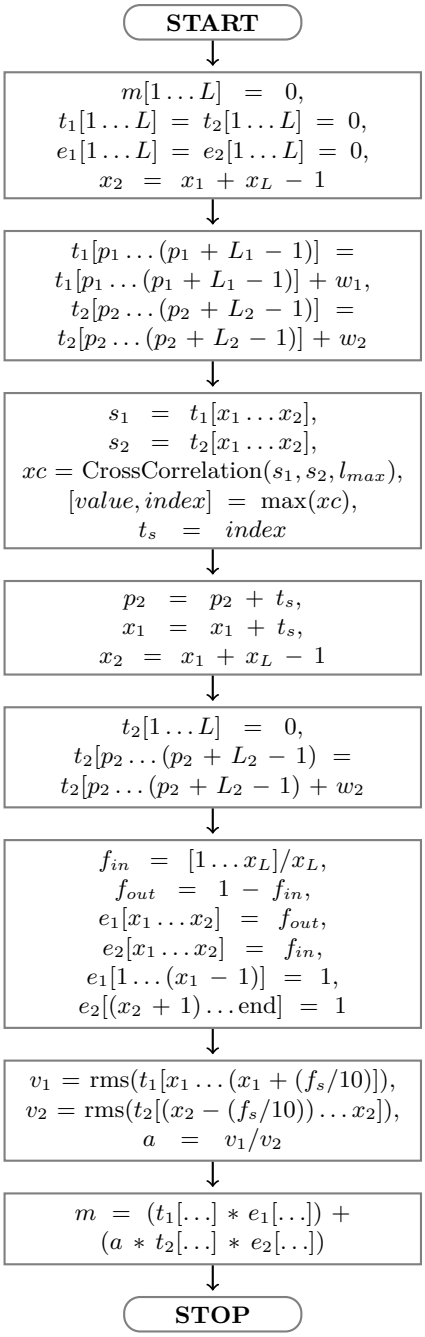


Figure 4.21. Concatenation of samples with phase-aligned crossfade

The idea is to time-shift the second sample in relation to the first one. Value of the shift is determined on a basis of cross-correlation between both signals in concatenated region – it is an abscissa of the first maximum of cross-correlation. The shift makes both signals phase-aligned, and allows them to be directly crossfaded with no amplitude distortion.

The adjustment is small enough not to impact the sensation of rhythm. In the majority of cases it does not exceed 1 ms, and even in the extreme situations[3] observed values only approached 10 ms. A value considered to be the fastest perceptual musical separation is 100 ms [339], therefore shift values applied by PA seem acceptable in comparison.

All stages of the actual concatenation procedure applied in PA synthesis are presented in Figure 4.21. In the initial stage empty arrays of required length are allocated: one for the mix, i.e. output signal, two for input tracks, containing positioned samples, and two for amplitude envelopes. Sound samples are positioned within their tracks on the basis of required duration of concatenated note. In each track a crossfade area is determined, and two respective track sections are isolated for calculation of cross-correlation. This provides the value of a time-shift, which is used to adjust crossfade area position, and to shift the second track. Corrected position of the crossfade area allows to calculate amplitude envelopes for both tracks. In the flowchart presented in Figure 4.21 envelopes are linear, though other can be applied as well. Signal amplitudes in concatenated samples does not have to be equal. Therefore in order to avoid producing an effect of short *crescendo* or *diminuendo* an amplitude ratio is calculated and applied as a correction in the final mix stage.

The procedure has a number of parameters that can be adjusted. Firstly, it is a matter of determining the extent of cross-fade area that would produce the least audible effect. Secondly, even with relative position of both samples determined by the rhythm, there is some space for choosing crossfade location. Thirdly, amplitude ratio can be determined on the basis of a variously sized neighbourhood of concatenation position. Considerations regarding usable values are discussed further in text.

4.5.2.2. Control of Duration

Both, PA and concatenative synthesis share an issue of note duration. Concatenated units contain sequences of pitches with inherent attributes of tempo and rhythm, which define time intervals between note onsets. Unlike them, conventional sampling has to deal with note duration only, and in contemporary implementations, not restricted by memory limitations, it is handled simply by recording long enough samples that only need to be faded out at some required point. A multi-pitch sample usually needs to continue to a next pitch even after duration of some of its inner notes has been altered. Therefore a simple fade-out does not suffice. One might try to apply a multisampling approach to note durations, and store various rhythmic sequences in various tempos. This however, is virtually infeasible. Even without it PA uses approximately 5000–6000 samples per instrument. With duration multisampling this number would increase by orders of magnitude, and even then duration would have to

[3]The worst case scenario considered was a shift of a half-period extent in signals with $f_0 = 50$ Hz.

be changed in relatively coarse steps. A more reasonable approach is to apply sample processing at the time of concatenation, and change duration as required.

Sample duration may be changed using various signal processing methods, many of which have been applied in implementations of concatenative synthesis, e.g. additive resynthesis [336], phase vocoder [190, 300], time granulation [273] or PSOLA [567]. These however, might stretch or compress fine temporal structures related to musical expression. In attempt to preserve expressive features present in recorded samples PA uses another approach, based on concatenation and insertion of additional samples.

Duration control technique utilised in PA is schematically presented in Figure 4.22. Both duration-related operations, shortening and lengthening, are based on concatenation, i.e. crossfading of samples after their phases have been aligned. Shortening is simpler of the two. It requires only the original sample, which is concatenated with its own copy, time-shifted backwards by a value of time difference between the initial and target duration. In essence, shortening removes part of a sustain region.

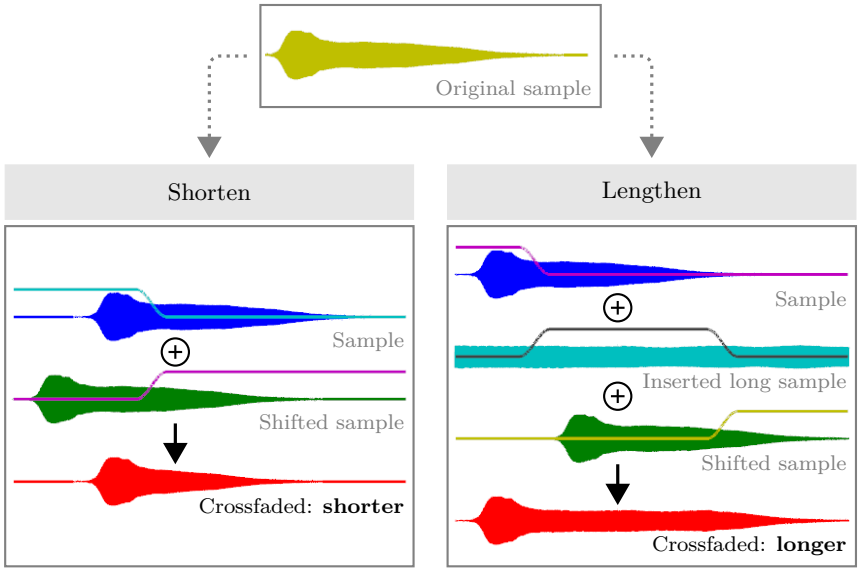


Figure 4.22. Duration control technique utilised in PA; in order to shorten a sample its copy is backwards shifted in time and concatenated with the original; lengthening uses an additional long sample which is concatenated with both, the original, and its forward time-shifted copy; concatenations involve crossfading, indicated by amplitude envelopes, and phase alignment

Lengthening is similar, only the copy of a sample is shifted forwards in time. This, however, leads to a problem. In cases of larger changes sample and its shifted copy will have no common sustain area available for concatenation: an attack phase of a copy can meet a release of an original, or there might even be a pause between samples. Here an additional sample is introduced. It is selected from single-pitch samples of long notes (Tab. 4.5), which are as long as possible, to perform using

a given instrument. The sample is inserted to fill-in the gap between concatenation areas of the original sample and its copy. Instead of one, two concatenations are performed. The original is concatenated with the inserted sample. The result of this operation has a very long sustain region, so that the copy of the initial sample can be shifted as required and both, the result of the first concatenation and the shifted copy, can be concatenated to produce longer note. Thus lengthening inserts an additional segment of sustain region.

For the purpose of simplicity, Figure 4.22 presents the duration control technique applied to a case of a conventional single-pitch sample. It is however a case of multi-pitch samples where the technique is essential. The fundamental feature is the fact that the only note segment affected by the technique is its sustain region, with remaining parts intact. Therefore neither initial attack, nor any of note transitions or final release are altered. This allows to reproduce virtually any sequence of notes with either constant or varying durations, while preserving original, fluent transitions between pitches.

4.5.2.3. Tempo and Rhythm

In case of detached notes control over rhythm and tempo in PA does not differ from a conventional sampling method. Single-pitch samples are placed in required positions on a time axis, and their duration is adjusted if required. Short articulations, like *staccato*, are already performed as short as possible, and often do not require an adjustment. In majority of the remaining cases long samples of special performance techniques are shortened.

Reproduction of rhythm and tempo using multi-pitch samples requires additional effort. The corpus of PA stores multi-pitch sequences with equal rhythmic values, but in different tempo variants (Tab. 4.6 and 4.7). Target tempo and rhythm are reproduced in a three stage process (Fig. 4.23).

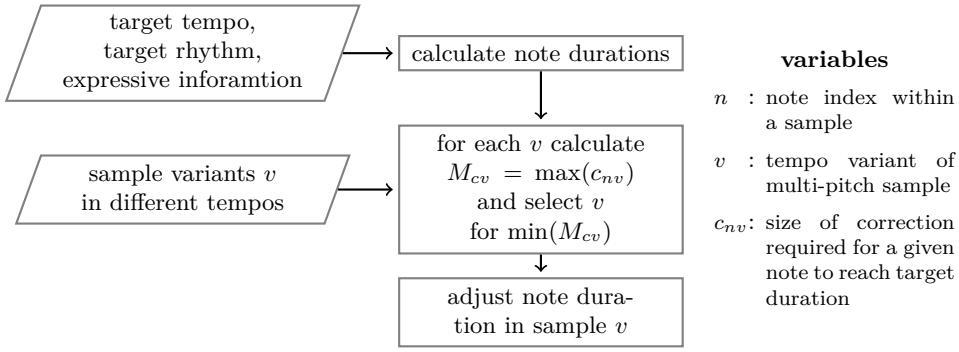


Figure 4.23. Reproduction of target rhythm and tempo using multi-pitch samples in PA

Initially, target durations for all notes within a phrase are calculated on the basis of required tempo, rhythmic values, and additional expressive information. Afterwards,

calculated durations are used to select a specific tempo variant of each multi-pitch sample matched to a phrase. The selection criterion is minimisation of the largest duration correction required for a particular sample. Finally, all note durations are adjusted during concatenation of samples into a phrase.

4.5.3. Performance Rules

Between an idea of a composer and a perception of a listener music undergoes a series of transformations. A score is only one of middle stages of this process. It is a specification and substantiation of a musical piece, but at a cost of a significant reduction of fine details. These details are filled in by a performer in a form of what is referred to as the expression.

A performance produced by a sequencer that literally reproduces a score differs from an expressive one by lack of certain kind of more or less subtle variations in performance parameters appearing in both, small and large scale. Such variations can affect note durations, fine tuning of pitch, dynamics, as well as articulation and note transitions. In most part they are not random, but rather originate from performer's interpretation of musical figures, structures, and forms, as well as understanding of course and gradation of musical tension and climaxes. Therefore they vary between performers, and there is never a common agreement as to how exactly a particular piece shall be performed.

Due to mostly indeterminate and fuzzy nature rules guiding expressive performance pose a serious problem for automatic music reproduction. They are hard to define and difficult to translate into a series of definite values that could be used to control a synthesizer. Yet, an extensive research has been carried out to establish quantitative description of this phenomenon, with the aim of introducing human-like expression to automatic performances.

There are two fundamentally different approaches to the problem [158]. The first one may be referred to as 'analysis through synthesis' [199]. In this approach the analysis starts from a model based on observations and experiences of seasoned, professional musicians. The model is applied to a set of examples that are evaluated by a board of listeners. This allows to tune or modify the model. Such procedure is iterated up to the point where the effect is satisfactory. The second approach may be referred to as 'analysis through measurement' [195]. It attempts to design a model by carrying out measurements of selected performance parameters in audio or video recordings. Video allows to supplement observations with parameters regarding player's movements and gestures. Besides the two approaches some researchers attempted to model expression by applying learning systems [82, 603, 604, 605] or fuzzy logic [83].

A model of expressive performance is usually translated into a set of *performance rules* which are precise enough to be applied to a music sequencer controlling a synthesizer. The rules match a specific context of a note, i.e. melodic, rhythmic, harmonic, or formal structures it belongs to, with defined variations of quantities characterising a performance.

The PA method adapts and applies a subset of performance rules referred to as the 'KTH rule system', developed by Bresin, Friberg and Sundberg [84, 195]. The system is extensively documented with a set of algorithms and software tools readily available in addition to the formulation of rules itself. KTH system divides rules into several groups related to pitch context, duration and meter context, intonation, phrasing, or synchronisation. The system has been designed to operate with conventional sequencers and synthesizers that assemble a performance out of single notes. In such case the rules are applied to signal amplitude, inter-onset duration (note spacing), offset to onset duration (note duration), amplitude of *vibrato*, or deviation from 12-TET tuning [195].

Due to differences between the PA and conventional synthesizers KTH rules had to be modified and applied selectively. Much of small scale expressive features is already present in recorded multi-pitch samples. It includes variations related to ascending or descending melody in scale sequences, or note transitions in intervals. Therefore some rules are obsolete. On the other hand, PA applies particular sample processing techniques that provide additional set of control parameters. Hence the set of quantities affected by the performance rules in PA consists of:

- pitch fine-tuning,
- signal amplitude,
- note duration,
- amplitude envelope,
- tempo envelope,
- and sample selection.

Quantities related to amplitude and duration are mentioned twice in the list. Once directly, and once bound in amplitude or tempo envelope. In both cases, separate and in envelope, control over the quantity is applied in similar manner, using the same mechanisms. The difference regards the area of influence. A single variant affects only a single note. An envelope shapes a larger musical structure, with more notes, at once. The last position in the list takes the advantage of sample-based synthesis multisampling feature. Particular performance techniques can be simply mapped to a specific sample variant. An appropriate selection of sample does not have to depend solely on the direct marking in score, but can also be affected by the context.

The rules are applied on the user request, and not all at the same time – some of them are mutually exclusive. Most of the rules allow to control the extent of their effect – from subtle to exaggerated, or inverted. Therefore a proper use is the matter of user's sense, though it is a simple matter to test various rule settings with the same score and select the best performance on the basis of an effect, and not an assumption.

The PA selection of performance rules is listed in Table 4.8 and Table 4.9. The first list includes a set of basic, general rules. The most fundamental is the 'phrasing' subset, and rules such as the phrase arch, the final *ritardando*, and the melody accent, which shall be applied in almost any performance. Rules from the second list are more discretionary. They partially overlap with expression already present in recorded samples, e.g. the 'intonation' subset, and when additionally included, they may actually harm the performance.

Table 4.8. A list of basic performance rules that can be applied in PA synthesis [158, 445], based on the KTH rule set [84] with modifications; abbreviated parameters are as follows: pitch fine-tuning (PT), signal amplitude (SA), note duration (ND), amplitude envelope (AE), tempo envelope (TE), and sample selection (SS)

Rule name	Comment	PT	SA	ND	AE	TE	SS
Phrasing							
Breath	Small pauses between phrases	–	–	+	–	–	+
Phrase arch	Arch-like change in tempo and signal amplitude	–	–	–	+	+	–
Final *ritardando*	Slow down in the end of a piece	–	–	–	–	+	–
High loud	Signal amplitude proportional to pitch	–	–	–	+	–	–
Metric accent	Emphasise metrical structure	–	–	–	+	+	–
Melody accent	Emphasise melody climax	–	–	–	+	+	–
Micro-level timing							
Duration contrast	Shorten short and lengthen long notes	–	–	–	–	+	–
Faster up	Increase tempo in ascending pitch sequence	–	–	–	–	+	–
Performance noise							
Noise control	Introduction of human-like inaccuracies	–	–	–	+	+	+
Basic articulation							
Legato and *staccato*	Two basic changes from normal articulation	–	–	–	–	–	+
Repetition	Slight separation of repeated notes	–	–	+	–	–	+

The rules are configured in a semi-automatic way, i.e. in most cases the synthesizer provides some default initial setting for the user to accept or modify. For instance, in the phrase arch rule amplitude and tempo arches are defined by nodes, initially set by the synthesizer on the basis of slurs and dynamics markings. Initial nodes can be moved or removed, and new nodes can be added at will. Using nodal coordinates PA calculates envelopes by application of shape-preserving interpolation with smooth first derivative using piece-wise cubic Hermite interpolating polynomial. The envelopes are used to control variations of respective parameters.

Table 4.9. A list of additional performance rules that can be applied in PA synthesis [158, 445], based on the KTH rule set [84] with modifications; abbreviated parameters are as follows: pitch fine-tuning (PT), signal amplitude (SA), note duration (ND), amplitude envelope (AE), tempo envelope (TE), and sample selection (SS)

Rule name	Comment	PT	SA	ND	AE	TE	SS
Tension							
Melodic charge	Emphasise distance from current chord	–	+	–	–	–	–
Harmonic charge	Emphasise distance from current key	–	+	–	–	–	–
Chromatic charge	Emphasise sequences with chromatic changes	–	–	–	+	–	–
Intonation							
High sharp	Stretch intervals according to size	+	–	–	–	–	–
Mixed intonation	Tuning to harmonic context in long chords, and to melodic context in fast sequences	+	–	–	–	–	–
Synchronisation							
Melodic sync	Synchronise notes in each voice to the 'collective voice' (with notes from all voices)	–	–	–	–	+	–

4.5.4. Phrase Envelopes

Some of fundamental performance rules, such as the phrase arch or final *ritardando*, translate specific musical context into dynamics and tempo envelopes that can span an entire phrase. These envelopes have to be considered already before concatenation of samples, so that concatenation parameters can be adjusted accordingly.

4.5.4.1. Dynamics Envelope

In live performance with acoustic instruments changes in dynamics require different playing technique. Therefore apart from signal level, dynamics alters articulation as well. Articulation, in turn, inherently affects sound timbre and attack phase of a note, or note transitions in case of connected articulations. Due to composite effect all but the simplest synthesizers supplement amplitude control in dynamics reproduction with other techniques, such as filtering or multisampling.

PA controls dynamics through continuous envelopes that can be shaped by single-note events, but which in case of most phrases are the effect of supplementing musical score markings with context-aware performance rules. Two mechanisms are utilised to simulate the composite effect. Alterations associated with articulation, which are particularly pronounced in case of larger changes in dynamics, are handled by multisam-

pling, i.e. selection of samples intentionally performed in given dynamics. Variations in sound level are handled by imposing amplitude envelope on assembled phrase.

The details of the dynamics reproduction are presented in Figure 4.24. The first mechanism is applied before concatenation of samples, during selection process. For each single- or multi-pitch sample selected on the basis of pitch and duration parameters, PA chooses a variant with recorded dynamics closer to the target. In early implementation two such variants are available: loud (f) and quiet (mp). Phrase sections that are to be performed in any of high dynamics levels, i.e. mf and above, are reproduced by the former variant, and the remaining ones – by the latter. Before concatenation sample levels are normalised using the root mean square (RMS) measure. Equalised samples are concatenated during phrase assembling. Resultant phrase has a globally flat amplitude envelope, but local, intra-sample expressive fluctuations are preserved.

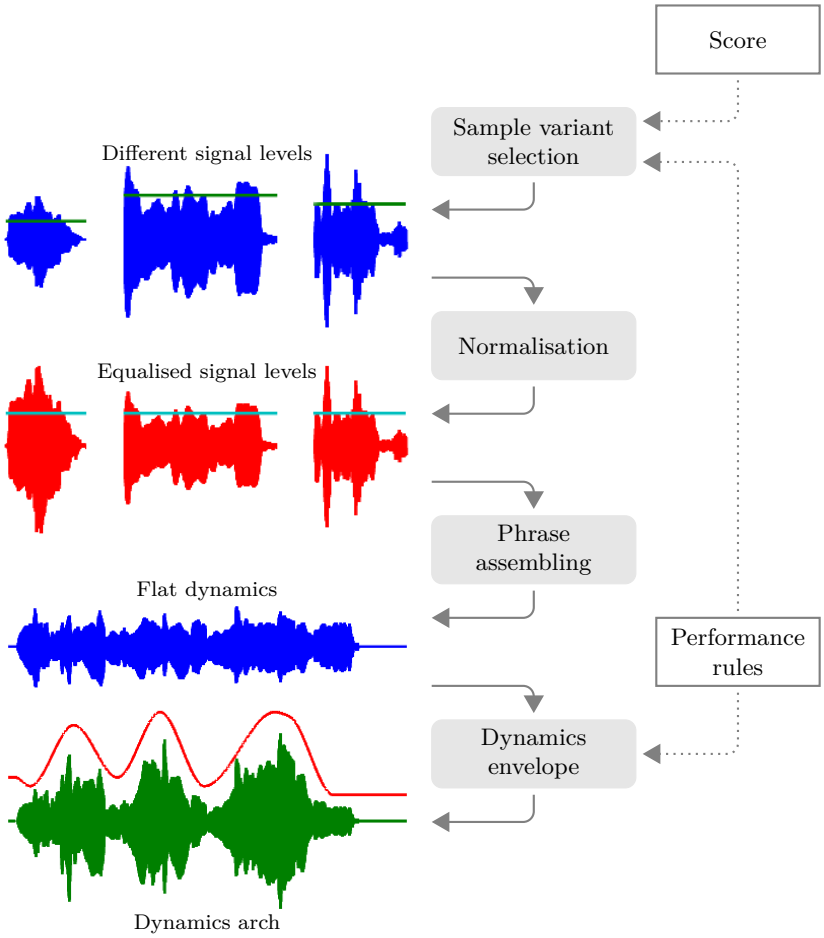


Figure 4.24. Reproduction of dynamics in PA

Further processing is applied to the entire phrase waveform. Envelopes are calculated on the basis of performance rules, which establish locations of nodal points associated with climaxes, slurs, etc. Envelope functions are obtained through shape-preserving interpolation with smooth first derivative using piece-wise cubic Hermite interpolating polynomial, and applied to modulate amplitude of phrase waveforms.

4.5.4.2. Tempo Envelope

Tempo markings in score are very limited in comparison to fine details present in live performances. It is often that musical pieces have only initial tempo marked, with no additional information throughout the whole piece. On the other hand, tempo envelope is one of more prominent expressive features and almost all types of musical context introduce some kind of characteristic irregularities into rhythmic sequences (Tab. 4.8). Therefore performance rules have a particularly important role of reconstructing tempo envelope comparable to that present in live performances.

Tempo envelope is obtained in a way similar to dynamics envelope. Number and locations of nodal points are determined using performance rules enabled by a user. Continuous envelope functions are obtained through shape-preserving interpolation with smooth first derivative using piece-wise cubic Hermite interpolating polynomial. However, unlike amplitude, which can be controlled in continuous manner, tempo is manifested through discrete note durations. Therefore envelopes are sampled on a rhythmic unit based interval, and respective duration corrections are cumulated for each subsequent note value (Fig. 4.25).

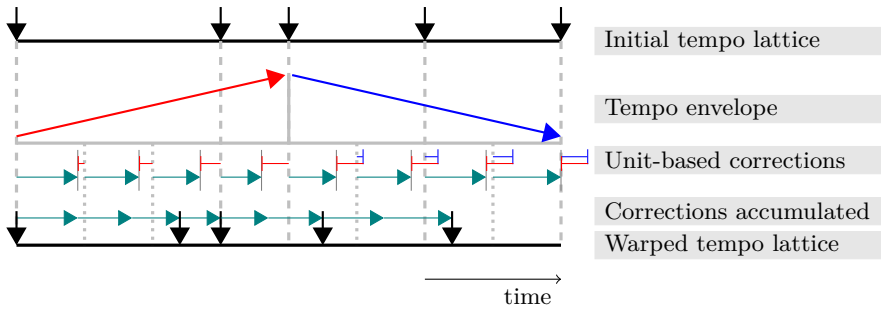


Figure 4.25. A schematic representation of calculation and accumulation of duration corrections according to a tempo envelope; in this particular case a simple linear envelope has been utilised; vertical black arrows represent note onsets

Corrections are implemented before assembling a phrase (Fig. 4.26). They are applied as shifts to elements of initial tempo lattice, calculated on the basis of regular tempo and note values. Warped lattice determines target durations of notes within multi-pitch samples as well as placement of samples on the time axis. Finally, required intra-sample cuts or insertions are applied during phrase assembling.

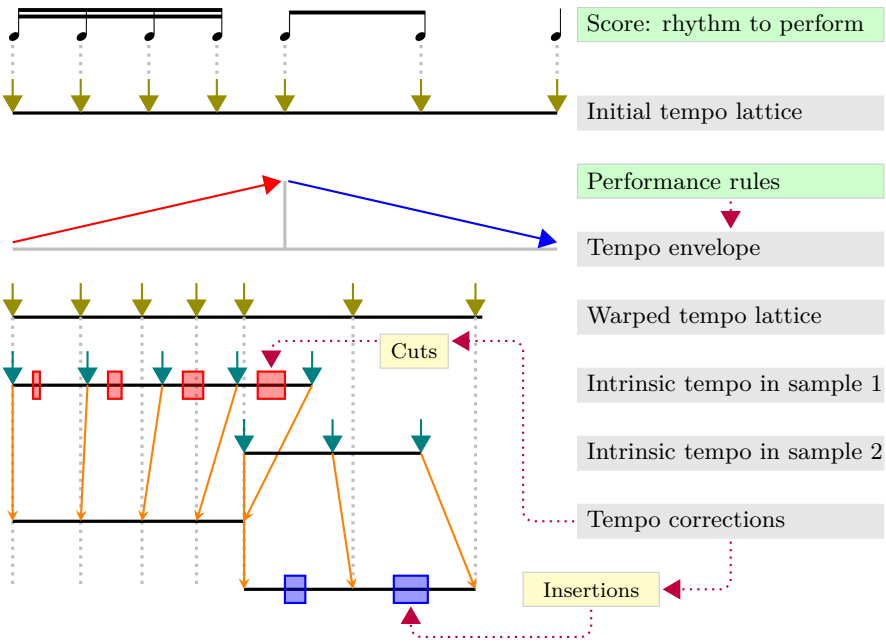


Figure 4.26. A schematic representation of tempo adjustment

4.6. Implementation

The phrase assembling sound synthesis method has been implemented in a proof-of-concept software synthesis system [446]. Main purpose of the program is to test different variants of particular algorithms and techniques, as well as to establish usable ranges for parameter values [159]. Therefore a modular design has been assumed in order to open the possibility for modifications and expansions. Due to its purpose the interface of the program has to be oriented not only towards a human user, but also towards automation and possibility to control it using other programs. Thus instead of graphical user interface the control mechanism involves textual configuration files, which enable interaction with both human, and supervising programs.

The program has been implemented using GNU Octave environment. Octave is an open source software, released under GNU General Public License. It is aimed at numerical computations and features a high-level, structured programming language. The language is interpreted with commands either invoked directly from a command line, or grouped in scripts for batch-processing. Octave environment is available for all major computer operating systems, including BSD systems, Linux distributions, MacOS, and Windows. There are attempts at running it under Android as well. It has a comprehensible documentation and can be easily integrated with other software.

Considering requirements of phrase assembling method implementation, Octave has a number of advantages. It is extensible, with a vast catalogue of available

modules, including signal processing related routines and algorithms, advanced text processing facilities needed for interpretation of score files, as well as convenient tools for data analysis and presentation. Moreover, its basic data structures are not single variables, but arrays in the form of vectors or matrices, with most of operations and functions working with arrays by default. As a consequence source code written in Octave language is clear and concise. As of disadvantages, two are important. Firstly, scripts are not standalone computer programs, but require Octave environment to run, though there is a possibility to produce binary executable files. Secondly, as a high-level interpreted language Octave is relatively slow. Both issues however, are of secondary concern in a proof-of-concept implementation.

4.6.1. Overall Program Design

The PA synthesis program is modular and has been designed as a set of Octave scripts. Some of them represent main system modules, while the remaining ones are supplementary and implement various data and signal processing algorithms and techniques, such as matching samples to a phrase, applying phase-aligned crossfade, or calculating the amplitude envelope.

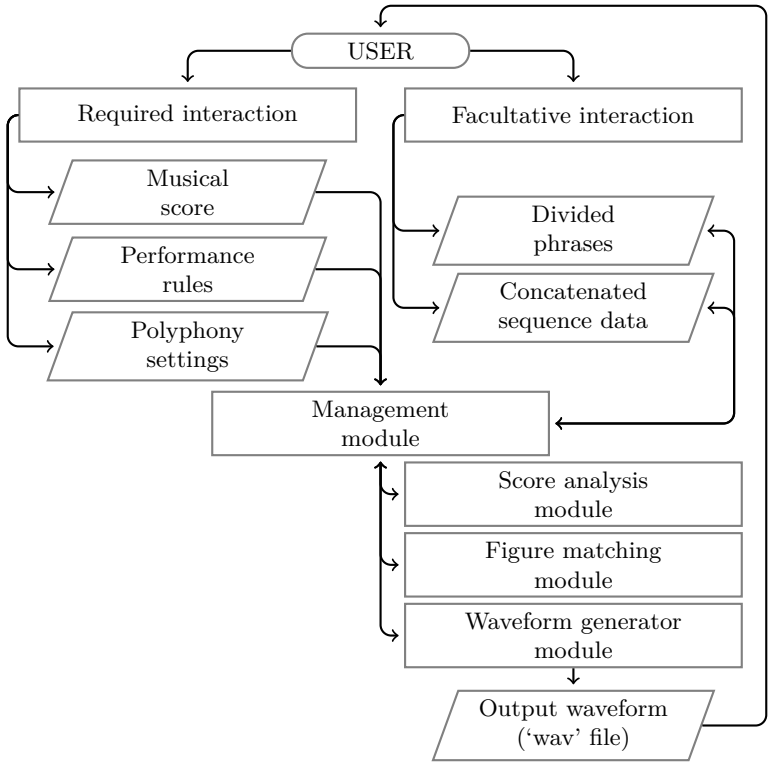


Figure 4.27. An overview of the implementation of the phrase assembling synthesis

Figure 4.27 presents a general overview of the PA method implementation. There are four main modules, three of which are processing modules and have specified tasks, while one referred to as the management module binds them all and serves as an interface between the synthesizer and the user or another supervising program. Out of processing modules the first executed is the score analysis module. It carries out a preliminary score analysis, i.e. reads score from a file and divides it into a sequence of phrases. Next one is the figure matching module. It is responsible for matching a sequence of samples to each phrase and determining overlapping notes. The last processing module is the waveform generator. It handles all of signal processing tasks and produces the output in a form of a 'wav' file.

User has to provide three sets of data. The first is the score file. The second consists performance rules settings, such as enabling of particular rules and values of parameters. The third is required in case of synthesizing more than one voice to be played simultaneously, and defines relations between voices. In addition to these sets user can interact in the middle of the synthesis process and modify intermediate synthesis parameters produced by processing modules. It includes division of score into phrases, as well as sequences of samples matched to phrases. Both can be reviewed and modified.

4.6.2. Modules

4.6.2.1. Score Analysis Module

The score analysis module has two tasks. The primary one is to divide a complete score into sections representing consecutive phrases. The secondary one is to perform pattern search in order to locate contexts where performance rules can be applied such as phrase arch areas, ascending or descending melodic progressions, local and global climaxes, changes of tempo or dynamics etc.

The analysis has several stages, presented in Figure 4.28. The initial step involves syntax analysis performed on the raw score data read from file. This allows to recognise various elements of musical notation. Some of these elements are irrelevant for the purpose of synthesis. Mark-ups regarding e.g. a page formatting or some textual information are discriminated in the following step. What remains is a solely musical information that can be interpreted and transformed into a performance. In order to avoid possible errors in further processing stages originating from score inconsistencies, the module checks parity of ties and slurs. This closes a series of initial steps.

Initial steps prepare score data for the main task – the search for phrase breaks. In the step that follows all the phrase breaking events are searched for, and a score is divided into sections representing single phrases. All single, detached notes outside of phrases are marked to be reproduced using conventional single-pitch samples. The last stage is the pattern search aimed at locating various contexts for performance rules. The output is a text file like the input, though it has a different structure that reflects phrases and highlights context patterns. The output of the score analysis module is also the input of the figure matching module.

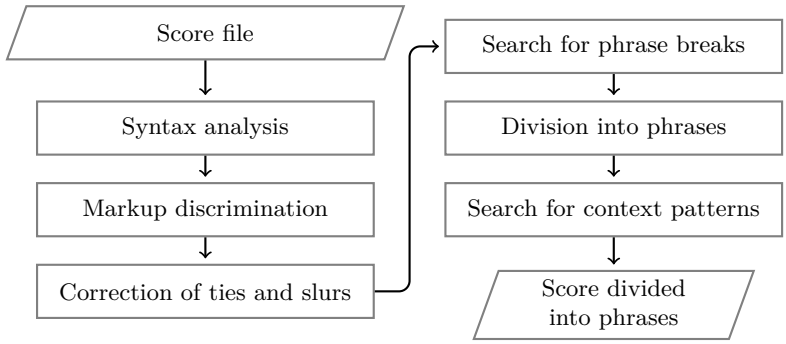


Figure 4.28. Execution flow diagram for the score analysis module

4.6.2.2. Figure Matching Module

In the figure matching module consecutive phrases are analysed and compared with contents of the PA corpus to find a matching sequence of samples (Fig. 4.29). A phrase is to be concatenated on 'overlapping' notes, common for adjacent samples. Due to the possibility of using whole as well as partial samples, the first and the last note of each sample used in a particular position within a sequence, i.e. its 'cutting spots', are included in sequence data. They represent concatenation locations.

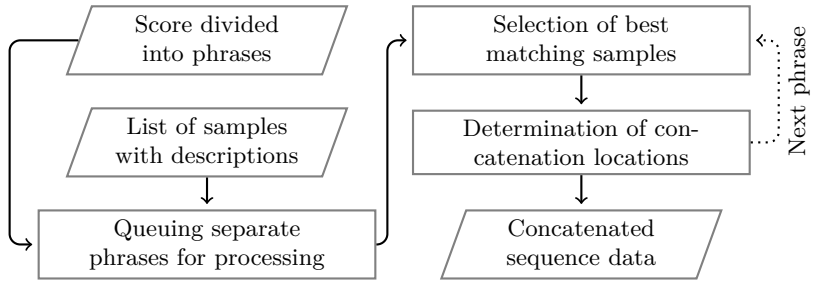


Figure 4.29. Execution flow diagram for the figure matching module

The output is written into a single text file that contains a whole synthesized piece, so it includes all phrases in order. The first section of the output file contains a list of unique sound samples that will be required for synthesis. The second section is a sequence of samples, which includes indices of respective samples and numbers representing the first and the last note within a sample. Each sample that is to be connected to the following one in a phrase is marked. Therefore lack of connection marking denotes the end of a phrase and beginning of a next one. Each sample has a target rhythm information attached. The file also contains the main tempo information, as well as two envelopes: one for tempo, and one for dynamics.

Each phrase is processed separately. In current implementation they are processed in sequence. However, in further implementations this step can be easily executed in

parallel to efficiently exploit multi-core processing elements and reduce processing time.

4.6.2.3. Waveform Generator Module

While score analysis and figure matching modules might by regarded as related to sequencer tasks, the actual sound synthesis is carried out in the waveform generator module. The module receives input from the figure matching module and uses it to control cutting, processing, and concatenation of sound samples from the corpus. The output is a waveform containing the instrument part. In case of polyphonic synthesis, output consists of several files containing parallel instrument parts that can be reproduced simultaneously or mixed into a single track. Tasks performed within the waveform generator module are presented in Figure 4.30 in a form of graph with directed edges illustrating precedence of operations.

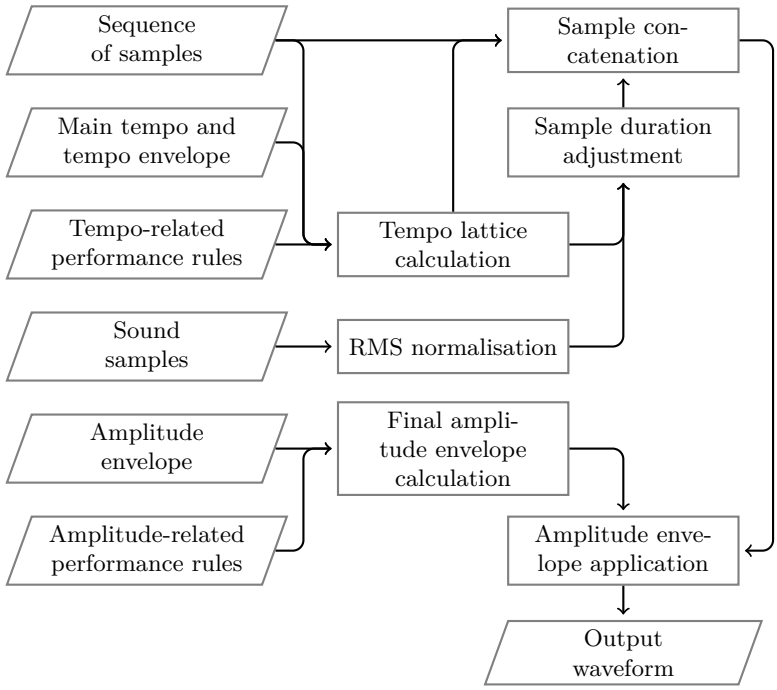


Figure 4.30. Execution flow diagram for the waveform generator module

Sound samples specified by the list in the input file are read, and their amplitudes are normalised. Positions of samples within output waveform are determined on the basis of the main tempo, the tempo envelope, and performance rules that can affect tempo. They constitute a tempo lattice. Using the lattice duration of notes within samples is adjusted to target duration through removing or inserting sections of sustain phase. The lattice and the sequence of samples from the input data define placement of adjusted samples in the output waveform. Positioned samples are con-

catenated on overlapping notes of each connected pair by applying the phase-aligned crossfade. Finally, assembled waveform is subjected to the amplitude envelope with regards to score markings as well as all amplitude-relevant performance rules.

In current implementation all operations are performed in sequence. However, as was the case of figure matching module, waveform generator consists of several stages that could be executed in parallel for improved computational efficiency in computers with multi-core processing elements. In the first place it is the processing of separate samples prior to assembling them into phrases, i.e. normalisation and duration adjustment. Additional parallelism is possible during concatenation, where each crossfade section can be calculated separately, although phase-adjustments have to be considered.

4.6.2.4. Management Module

The management module coordinates operation of the three processing modules (Fig. 4.31) and interacts with a user or a supervising software through configuration files. It controls launching of modules in appropriate order and assures correct data interchange by receiving, checking and sending required information at each stage of analysis and synthesis.

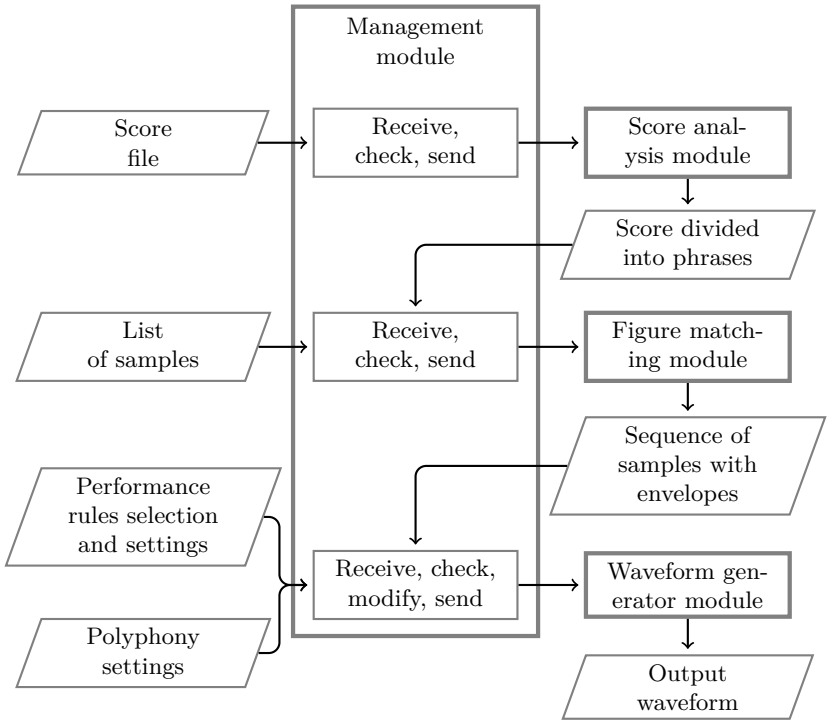


Figure 4.31. Execution flow diagram for the management module

If the synthesizer is used in polyphonic mode, where additional performance rules need to be applied, the management module coordinates and synchronises voice parameters. In this case not only it relays data between the processing modules, but it can modify relayed data as well. Modifications primarily affect tempo envelope, which needs to be coherent in all voices playing together. Amplitude envelopes are more independent, but some modifications are still required – voice amplitudes need to be uniformed in contexts such as global climaxes or endings.

4.6.3. Program Parameters Adjustments

While the PA method is complete in terms of type and order of performed operations and applied algorithms, a final operational implementation requires a further, experimental phase of development, with the objective of determining operational parameters for consecutive processing stages. A number of initial experiments have been carried out to establish values for the most fundamental procedures that allow to produce undistorted and realistic performances [159, 445].

Majority of parameters which values need to be established regards the details of concatenation operation, and particularly location and length of concatenation region (Fig. 4.32).

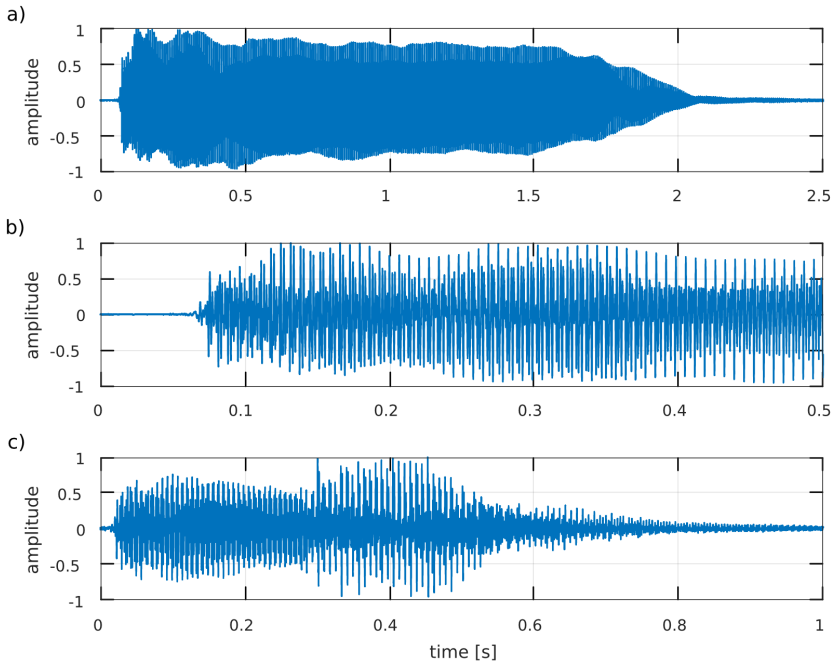


Figure 4.32. Waveform of a single-pitch bassoon sample (a), a zoom on its attack and decay region (b), and a two-pitch bassoon sample (c); sustain regions can be determined only approximately

The PA corpus consists of recordings of live performers using acoustic instruments. While it is advantageous for reproduction of expression, it is also a source of irregularities. Frequency-related, amplitude-related, and temporal parameters of samples as well as their spectral characteristics are fluent, change gradually and usually non-monotonically, hampering precise segmentation. As a consequence contents of samples demonstrate no exact pitch transition moment, and no clear boundaries dividing particular notes into phases. A clear sustain phase is an idealisation and may be encountered in purely synthetic instruments, where a line-segment envelope approximation has been utilised to control amplitude. In sound of real instruments phases like attack, decay, sustain or release are less distinct, if present at all. Therefore segmentation, required by the PA method, has to be more or less arbitrary, with no clear point of origin that could serve as an anchor for positioning concatenation region (Fig. 4.32).

Other concerns regard control of duration, which has to be extensively utilised throughout entire synthesized waveform, as well as form and details of amplitude envelope and extent of tempo envelope, both crucial for expressive performance. Moreover, an influence of additional background noise level has been studied, as well as a number of minor adjustments and choices.

4.6.3.1. Listening Tests – Phase I

In the initial round of tests [159] a number of performances has been produced using the PA synthesizer with different parameter settings on the basis of two musical excerpts presented in Figure 4.33. A part of flute from Dvořák's symphony includes expressive *legato* phrases with fluent note transitions occurring on various intervals and note durations. They are preceded by a few detached notes in the initial section. The last phrase has marked gradual dynamics changes, *crescendo* and *diminuendo*. This is a typical target music for the PA synthesis. A bassoon part from Mozart's symphony is different. It consists of mostly detached notes with only one short phrase before the end that has fluent transitions. The excerpt includes large duration contrasts, a few fast groups, several notes with short articulation, and one special performance technique – a trill in the fourth bar.

Listeners were comparing pairs of performance variants that differed in selected parameter or feature. They were instructed to choose a variant that was closer to live performance within each pair. A single performance variant is referred to as a test sample. Details of the test setting are presented in Table 4.10.

Table 4.11 presents detailed information regarding parameter changes introduced into test samples. Changes affect main tempo of performance, length of region that is cross-faded between adjacent samples, section of note considered its beginning and end, as well as presence of phrase arch. Note beginning and end are two areas that are excluded from crossfading, i.e. the remaining middle section is considered a sustain region, allowed to be crossfaded. Phrase arch contains two separate arches, for dynamics and tempo, that can be individually enabled or disabled. Out of 42 test samples 21 are performances of Dvořák, and 21 of Mozart excerpt. For each excerpt there is one variant assumed as a reference (No. 1 and 22), with parameter values established on the basis of preliminary tests.

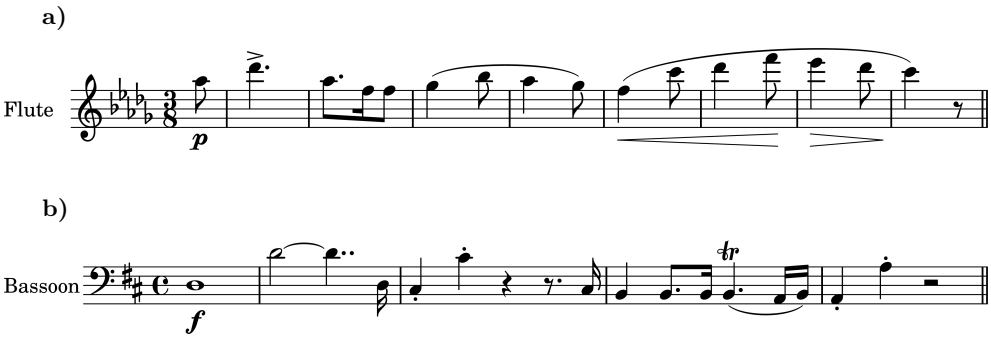


Figure 4.33. Two excerpts of instrument parts from symphonic music selected for tests: a) A. Dvořák Symphony No. 5 in F major Op. 76, Movement III – Trio, bars 285–292; b) W.A. Mozart Symphony No. 35 in D major K. 385, Movement I, bars 1–5

Table 4.10. The setting in the phase I of the listening tests

Test participants	15 listeners including 10 experienced university ear training teachers and 5 PhD student orchestra conductors
Procedure	Fixed sequence of pairs of test samples presented; forced choice of one test sample from each pair; playback and repetition on demand; procedure controlled automatically by a computer software
Listening conditions	Closed studio headphones (Beyerdynamic DT 770 Pro); sound level set individually per listener; listeners allowed to take breaks at any moment during the test
Signal presentation	Diotic: 1-channel/mono samples presented simultaneously to each ear
Test duration	50–120 minutes, in most cases approximately 70 minutes

Only one parameter was changed from reference value at a time. Each of tempo variants were included in a test sequence only once. The remaining parameter variants appeared twice each, with changed order, i.e. reference-first, and reference-second. In total, each listener evaluated 80 pairs of samples, and the order of presentations was always the same. Samples were grouped according to tested parameter.

The results are presented in Figures 4.34 and 4.35. Charts show percentage of choices of either variant. A difference can be assumed significant (with a level of significance $\alpha = 0.05$ of a two-sided exact binomial test) if it is no lesser then 60 percentage points in case of tempo variant comparison and 40 percentage points in remaining cases.

When choosing between tempo variants listeners opted against the fastest performances, i.e. 117 BPM in case of Dvořák and 200 BPM in case of Mozart. The fastest variant was rejected not only when compared to an average one, but when compared to a fast, albeit slower one. Neither crossfade length variant was preferred in case of

Mozart, with mostly detached notes, but in Dvořák, with its longer phrases, listeners avoided short variants, i.e. up to 28 ms.

Table 4.11. Parameters of test samples used in the phase I of the listening tests; abbreviated parameters are: tempo [BPM] (T), note beginning [ms], area excluded from cross-fading (B), note ending [ms], area excluded from cross-fading (E), crossfade length [ms] (X), presence of phrase arch – dynamics/tempo (P); 'R' represents values that are the same as in the reference

Sample variants	T	B	E	X	P	No.
Dvořák						
Reference	100	68	23	113	1/1	1
Tempo	60, 80, 117	R	R	R	R	2–4
Cross-fade length	R	R	R	6, 11, 28, 57, 85, 170, 283	R	5–11
Note beginning length	R	23, 45, 91, 113	R	R	R	12–15
Note ending length	R	R	11, 45, 68	R	R	16–18
Phrase arch presence	R	R	R	R	0/0, 0/1, 1/0	19–21
Mozart						
Reference	150	68	23	113	0/0	22
Tempo	125, 175, 200	R	R	R	R	23–25
Cross-fade length	R	R	R	6, 11, 28, 57, 85, 170, 283	R	26–32
Note beginning length	R	23, 45, 91, 113	R	R	R	33–36
Note ending length	R	R	11, 45, 68	R	R	37–39
Phrase arch presence	R	R	R	R	0/1, 1/0, 1/1	40–42

Similarly, and possibly due to same reasons, no significant preference was observed in Mozart piece for the length of note beginning and ending. In case of Dvořák listeners preferred note beginning lasting 68 ms over both, longer and shorter values, and note

ending lasting 23 ms. Longer endings were avoided, but there is no clear preference for 23 ms over shorter values. Interestingly, no preference was observed for either variant of phrase arch in Dvořák piece, while a significant preference against the use of tempo arch was observed in Mozart piece.

The results allowed to establish approximate range of working values for concatenation-related parameters such as crossfade length and concatenation-available area limited by note beginning and note ending lengths. They also reveal a few places for improvement. Firstly, problems with the fastest tempo indicate that either the corpus should contain a larger range of recorded tempos instead of current two (60 and 120 BPM), or the concatenation-available area should be determined more precisely. Secondly, phrase arch implementation used to control amplitude, and particularly tempo envelope, requires adjustments.

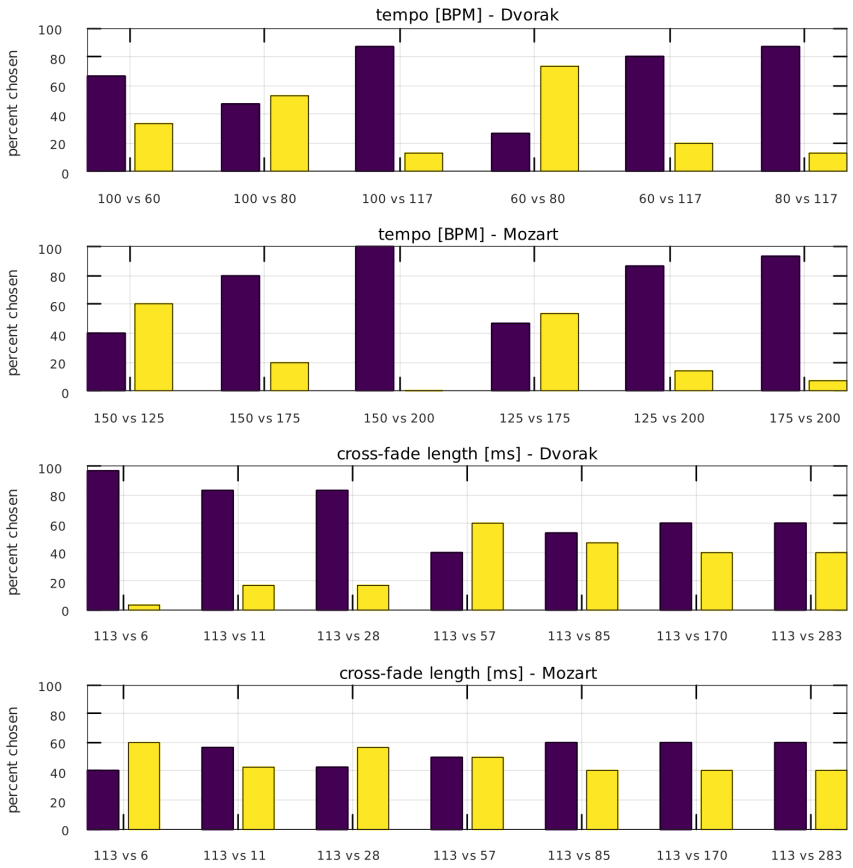


Figure 4.34. Results of the first phase of the listening tests – a comparison of tempo variants and crossfade lengths, after Delekta, Spałek and Pluta [159]

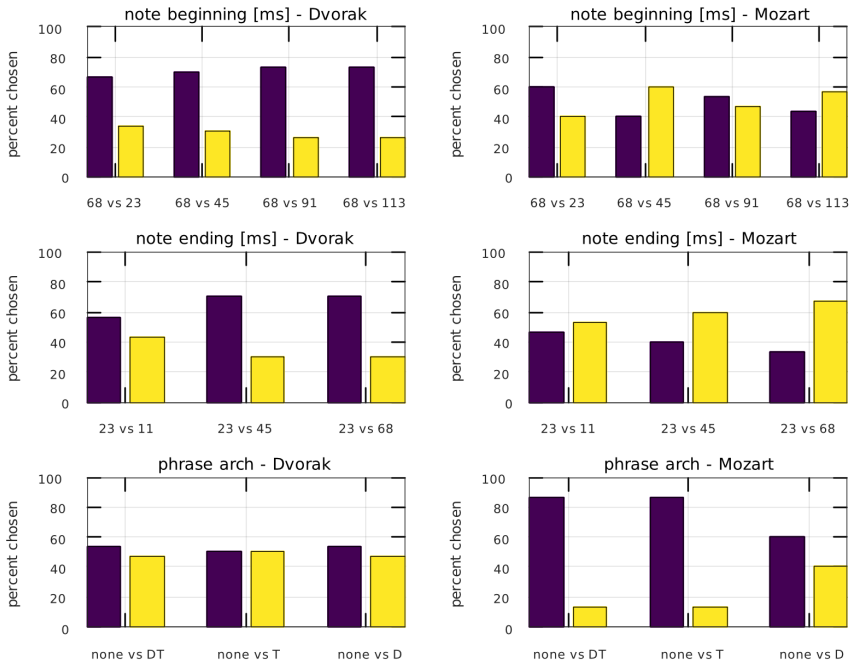


Figure 4.35. Results of the first phase of the listening tests – a comparison of note beginning lengths, note ending lentgths, and presence of phrase arch in dynamics (D) and tempo (T), after Delekta, Spałek and Pluta [159]

Due to specific background of listeners, all of which were experienced and highly skilled professional musicians, they were asked for a feedback. Three areas were pointed out. Many opinions regarded articulation and accentuation, however they were mutually contradictory. The second area of consideration regarded long notes, which in evaluated performances were perceived as idle in comparison to how they are performed by musicians. This points towards introducing an additional performance rule, and has been considered for further study. The last area was of a different nature. Listeners perceived as unnatural situation a perfect silence occurring between notes, while instead they expected some kind of background noise. It has been considered as a necessary addition for the waveform generator module.

4.6.3.2. Listening Tests – Phase II

Results obtained in the first phase of tests allowed to improve the PA synthesis implementation by introducing modifications into selected parts of program and by adjusting its parameters. The improvements included addition of background noise to the synthesized sound, altering duration and amplitude envelopes applied in fade sections, modification of tempo and amplitude envelopes used in phrase arches with addition of emphasis rule for long notes, and fine tuning of pitches to 12-TET.

The test was carried out using Dvořák piece from the first test phase due to its more conclusive results compared to Mozart piece. Changes introduced to the program were divided into six categories, as described in Table 4.12.

Table 4.12. Description of changes introduced before the phase II of the tests and related sample variants used in the tests; symbols in the left column correspond with symbols on the X-axis in Figure 4.36

Symbol	Introduced change	Sample variants
noise bg	Background noise presence and level; noise recorded in a recording studio; noise level relative to the signal level, RMS	1: no background noise (old) 2: noise level -45 dB 3: noise level -39 dB 4: noise level -33 dB
fade env	Envelopes of fade-in, fade-out, and crossfade segments	1: cosine ('s-curve') 2: linear (old)
fade dur	Fade-in and fade-out segment duration	1: in/out 20/80 ms (old) 2: 120/60 ms (softer onsets)
tmp arch	Presence of the new tempo phrase arch; compared to the method applied in phase I the new one introduces smaller (up to 3%) and smoother tempo changes; a continuous envelope is calculated per rhythmic unit, and not per note, as was the case in phase I	1: tempo arch present 2: no tempo variations
dyn arch	Presence of the new amplitude phrase arch; compared to the method applied in phase I the new one is smoother, calculated on the basis of smaller number of nodal values, while the old one used one node per note; an additional emphasis on long notes is introduced in a form of *crescendo* and subsequent *diminuendo*	1: dynamics arch present 2: no variations in dynamics
tuning	Fine tuning of pitches to 12-TET system; without tuning minute pitch discontinuities may occur on some crossfades; it is avoided by tuning, although such procedure removes some expressive information, such as tuning of leading notes or special intervals	1: pitches tuned to 12-TET 2: originally recorded tuning (old)
all	All changes introduced together; improvements are represented by the following choices: noise bg 2, fade env 1, fade dur 2, tmp arch 1, dyn arch 1, tuning 1; the old variant is represented by: noise bg 1, fade env 2, fade dur 1, tmp arch and dyn arch in the from from phase I, tuning 2	1: all improvements together 2: old (as in phase I)

Effects of changes were individually compared to the previous program version. Additionally, seventh category grouped all improvements together. Within each category a pair of samples was prepared, one for the 'old', and one for the 'new' variant. In case of background noise, which amount could be graded, four separate samples were prepared. Test settings are described in Table 4.13.

Table 4.13. The setting in the phase II of the listening tests

Test participants	23 listeners, including violin, composition, and conducting students, along with ear training teachers from the Academy of Music in Kraków, as well as sound engineers and acousticians
Procedure	Presentation of a sequence of seven sample sets, as described in Table 4.12; forced choice of the best sounding sample from each set; playback and repetition on demand
Listening conditions	Speakers (near-field studio monitors) or closed studio headphones; sound level set individually per listener; listeners allowed to take breaks at any moment during the test
Signal presentation	Diotic: 1-channel/mono samples presented simultaneously to each ear

The number of test participants does not allow to deem results, which are presented in Figure 4.36, as statistically significant at a confidence level of 0.95 [445]. The last set ('all') representing implementation changes applied altogether and confronted with the old variant is on the borderline, and becomes statistically significant at a confidence level of 0.9.

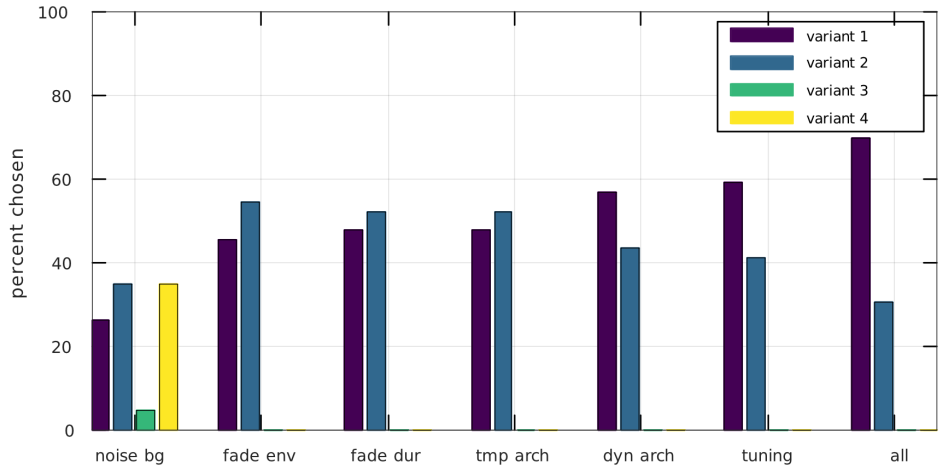


Figure 4.36. Results of the second phase of the listening tests – a comparison between variants for selected performance features, after Pluta, Spałek and Delekta [445]; symbols of changed features appearing on the X-axis are described in Table 4.12

Interestingly, results from the last set ('all') reveal listeners' preference towards the new variant, yet none of the changes seems to be responsible for such preference while applied alone. The only other set that might provide significant result is the one designated as 'noise bg', but it would require grouping of all background noise level variants into the one 'noise' case compared to the variant with no background noise. Therefore even though it is difficult to pinpoint exact improvement responsible for attracting preference of listeners, it allows to set the entirely new 'all' variant as a new default setting for the synthesizer implementation.

4.6.4. Evaluation

The PA method of sound synthesis has been designed mainly as an alternative to sampling synthesis for the purpose of automatic reproduction of musical score. Even though its implementation is still a work in progress and should not be considered ready for immediate application, a small, less formal listening test has been carried out to evaluate its current state in comparison with a commercial sampler.

Two sound samples have been produced, each one being a reproduction of the same Dvořák piece that has been used in previous tests (Fig. 4.33a). One has been reproduced by the PA implementation with all described improvements, and the other one has been reproduced from a standard MIDI file by a sampler – the 70 GB version of Independence Premium Library included in the Samplitude Pro X Suite DAW. 17 listeners participated in the test – musicians, acousticians, and sound engineers. Test setting from the phase II of previous listening tests has been applied. Listeners were asked to choose a better sounding variant.

8 listeners have chosen the PA variant, and 9 have chosen a sampler. PA method might therefore be considered as an alternative to sampling, although its implementation clearly requires further refinement. As a more complex and more constraint method PA should not only match, but surpass performance of sampler in order to become its rational alternative.

4.7. Concluding Remarks

Automatic reproduction of musical score is one of key areas of sound synthesis applications. Due to various shortcomings of synthesis methods currently utilised there is still a place for alternatives. The phrase assembling method aims to be one of such alternatives with a number of advantages over sampling, which is currently the most common choice, as well as over the concatenative synthesis, recently acquiring a growing acceptance. The formulation of the PA method along with details regarding underlying techniques and algorithms has been presented. Conducted tests indicated areas of possible improvement, so that the PA could become a viable alternative for other sample-based methods. Apart from refining the implementation, it is also possible to expand method capabilities to allow its wider application.

4.7.1. Issues and Necessary Improvements

The most fundamental issue of the PA method, and the one that is a source of the most noticeable distortions in the output signal, is the process of segmentation of multi-pitch samples. In current implementation sustain regions are determined approximately, based on a set of generalised values obtained in listening tests. However, only a precise segmentation into regions representing subsequent pitches and finding sustain regions within each note will allow to avoid signal discontinuities. Trials and application of descriptors adapted from the field of MIR, either based on a combination of fundamental frequency, amplitude, tonal/noise distinction, and transient detection, or derived from higher level features, shall provide a more accurate and robust solution [169].

Even though corpus of sound samples utilised by the PA is extensive, tests have indicated that there is a need for more fine-grained tempo variants in order to reduce larger modifications of note durations. This in turn, shall be accompanied by redesign of the corpus in order to keep its size from expanding.

The final issue is the extent of the corpus itself, and specifically amount of work required to prepare it. Apart from usual recording and processing, PA samples have to be further processed and segmented. Even though it is a one-time operation per corpus, in current state of development it still requires a considerable amount of manual work or – in case of automated tasks, such as pitch segmentation – a human supervision. Again, MIR assisted with techniques facilitated by contemporary sound recording and processing tools might provide means for further automation to reduce human interaction during the entire process. In turn, production of larger corpora might become attainable. On the side of a recorded musician it shall speed up the recording if the samples instead of being played separately are formed into one or several study-like quasi-musical pieces containing all required figures and techniques. This however would contribute to more sample processing, so it can be carried out only when an efficient and reliable automation has been developed.

4.7.2. Further Development

The PA method has been developed with a clear, yet relatively narrow objective: to reproduce scores of orchestral parts of wind instruments with particular attention to fluent phrases and expressive features. This sole objective can be expanded in various directions, opening a number of paths for further development. The list of such directions includes:

- expansion into other instrument groups,
- tuning and automation of performance rules,
- selectable simulation of various performance styles,
- real-time performance,
- using elements of PA in hybrids with other synthesis methods,
- applying elements of PA outside the area of sound synthesis.

Wind instruments are particularly well suited for simulation using the PA method. In case of other instruments or groups of instruments benefits of the method are less apparent, yet there may be a merit in synthesizing the whole orchestra or other ensemble using the same method[4]. Therefore it is possible to expand the PA corpus to string instrument groups and solo strings, as well as percussion instruments, grand piano, harpsichord, organ, etc. The corpus however, is not the only modification required. Due to various performance techniques in different instrument groups the synthesis engine might require adjustments in concatenation method, and will definitely require changes in performance rules.

PA adapted a subset of performance rules from the KTH rule system even though these rules were originally developed for conventional synthesis methods, where every note is a separate event with the same control parameters. Rule settings and parameters were adjusted to PA in a series of trial and error experiments. Development in this area may include testing of different rule sets, as well as complete removal of the necessity to adjust rules manually through better automatic detection of various levels of musical structures and contexts.

Current implementation aims at reproduction of orchestral parts of musical pieces from specific periods. It does not attempt to handle older or modern music as well as solo performances due to much broader spectrum of styles and performance features required. Yet, PA may be applied to simulate different musical styles and periods as well. It would require expansion of the sample corpus and definition of much larger rule set. Moreover, the style should not be detected automatically, but rather selected by the user as a *preset*. In this way the same piece could be performed differently with distinctive expressive features, without a need to set all required performance rules separately.

The PA is in its roots a non real-time synthesis method. One of its main features is the score analysis stage, which requires access to the entire reproduced piece in order to select fitting multi-pitch samples and to search for contexts for performance rules. Though, as in case of concatenative synthesis, which is another non real-time method with score analysis stage, there is a possibility to develop a real-time variant of the PA method. Actually, there may be two different ways to introduce a real-time control over a PA performance: the **performer mode** and the **conductor mode**. The first one would allow to play anything by skipping almost all performance rules which require a broader context, and by using only single note and interval samples. As in case of monophonic synthesizers, there would be an initial distinction whether a note played is detached or attached to the previous one[5]. In case of the latter an interval sample would be concatenated to the currently sounding one, and otherwise a single note sample would be reproduced. In the conductor mode the user would not be able to play anything, but instead would control performance of a pre-analysed score, with a fixed sequence of samples determined beforehand, and performance rules initially set. Depending on the user interaction parameters such as tempo, dynamics,

[4]Even with separate vibrators for each produced pitch and lack of fluent note transitions, attack segments may vary depending on the musical context, which is handled by the PA method.

[5]Such detection is typically based on the overlapping of pressed controller keys, i.e. whether before the next key pressed the previous one has been released.

and selected controls of performance rules would be altered during reproduction. Such mode would be a useful training aid for conducting students, as well as an interesting entertainment tool allowing a non-musician to control a virtual musician or ensemble.

Sound synthesizers, especially in commercial applications, rarely implement a single, pure synthesis method. Instead it is common to create hybrids of various methods to attenuate their deficiencies and complement methods selected as a base with fitting elements from others, e.g. a sampler as a signal source with subtractive modifiers. The most interesting hybrids of PA would be with additive and subtractive methods. Additive resynthesis or subtractive phase vocoder could be used to assure a continuity of two overlapped samples in concatenation region. Moreover, subtractive modifiers or signal modulators could enhance PA control capabilities and allow to apply subtler performance rules. This however could contradict expressive features already present in recorded samples. Other way around, sampling synthesis could apply PA interval samples and concatenation to produce fluent pitch transitions.

Finally, PA method can be taken apart and various combinations of its techniques and algorithms can be applied in sound engineering, processing, or research. One of the most obvious is the study of performance rules. Other could be a research of various music-related perceptual phenomena, such as pitch intonation [439]. It would require using a PA implementation as a basis, with required control capabilities added, for instance, an arbitrary pitch shifting. In an entirely different approach, analysis unit of the PA and its performance rules could be used as a basis for robotic musical performance in connection with real acoustic instruments [555, 556].

5. Infeasible Instruments: a Novel Means for Music Performance

5.1. Synthesis Methods for Music Performance

A sound synthesizer may be considered a source of musical sounds non different than any musical instrument. It has been reflected in naming conventions of contemporary audio software, where a synthesizer implemented as a computer program in the form of a plug-in is referred to as a *virtual instrument*. However, synthesizers have a distinct feature uncommon for most of other instruments – they allow to control timbre of a sound in a more fundamental, broader, yet also more precise manner. In effect they are able to imitate other sound sources.

The ability to imitate other instruments led to some degree of specification among synthesis methods and synthesizers. One group of synthesizers attempts to accurately recreate sound and control behaviour of existing instruments. The other group aims at inventing new sounds. The latter group utilises various techniques, or even abstract algorithms, in a process that is often dissimilar to the sound generation in real instruments.

Clearly, only synthesizers from the first group may be utilised for music reproduction purposes if the score to be reproduced is written for a conventional instrument, unless some instrumentation experiment is considered. This scenario has been discussed in previous chapters. In this chapter the point of interest is a live music performance. Here synthesizers from both groups are utilised, but with slightly different goals. The first group serves as a replacement for instruments or ensembles that would be otherwise problematic to perform with, under given conditions. Synthesizers from the second group are regarded as distinct instruments. Here all the control capabilities as well as novelty and originality of produced sounds play the prominent role.

5.1.1. Control and Timbre Capabilities

A timbre of sound generated by a synthesizer used for a real-time performance may be controlled in two different manners. In the first one, main part of the process is carried out prior to the performance, and only limited adjustments are possible while performing. The reason is either complexity of control procedure or large number of parameters to control. In the second manner most part of timbre control is carried out during the performance using dedicated controllers. This however requires a synthesizer to facilitate a limited set of timbre control parameters, so that each of them might be mapped onto a controller. Moreover, these parameters need to have a clearly audible effect, so that they could be used as elements of a performance. While in general the first manner might allow to obtain more complex, elaborate sounds, the second one introduces a feature characteristic for the most expressive instruments, such as violin or human voice, i.e. deep, performance-bound timbre variability.

Among the most widespread synthesis methods capable of real-time performance, listed in Table 5.1, majority shares the ability to control timbre of sound during musical performance. Two notable exceptions are additive synthesis and sampling, each due to a different reason. While additive synthesis allows for very detailed definition of timbre, it achieves it by tending to an immense set of parameters. Values of most parameters alone have only a slight impact on the timbre. A performer attempting to control timbre during a performance has to resort to a mechanism that will group parameters and allow to control their larger subsets with single controllers. Timbre control in sampling is essentially very simple, though it cannot be used to create any new sound. Even after supplementing a sampler with a subtractive block of signal modifiers, timbre is predominantly determined by features of an underlying sound sample.

Two of the remaining methods, i.e. subtractive and FM, are considered the most capable for real-time timbre control with audible, distinct effect, and relative ease of obtaining new sounds. Here, a new problem might arise, depending on the experience of the performer. It regards the meaning of parameters and their correspondence to well-understood timbre features. Basic parameters of subtractive method, a filter cut-off or centre frequency, are intuitive. However in more elaborate filters effect of some parameters may be obscure. The same applies to FM parameters. As a result control of these parameters might be carried out with less purpose, and more through a trial and error method.

An interesting case is the physical modelling synthesis. Its timbre control mechanism stands out from the remaining methods. Here, it is not a sound that is directly controlled, but a model of instrument and its excitation. Parameters represent physical features of the instrument modelled, therefore are inherently intuitive, particularly in connection with a controller that mimics control mechanisms of the simulated instrument. In such situation performer can utilise skills acquired with conventional instrument. However, in its main form the physical modelling is not intended to create new sounds. Instead, it attempts to recreate some physical original. Therefore it is rarely considered a method of choice for the purpose of experimenting with novel timbres. It is only a matter of changing a model, though. Either by modifying models

of instruments, or by choosing entirely different objects as sound sources, one might use physical modelling not to replicate, but to design new sounds.

Table 5.1. Feasibility of real-time sound timbre control in selected real-time synthesis methods

Method	Timbre control	Real-time feasibility
Direct methods		
Additive	Amplitude and frequency envelopes for each partial	Requires intermediate solutions to manage groups of partials
Subtractive	Selection of base waveform, filter cut-off or centre frequency, remaining filter characteristics	Feasible
Wavetable	Selection of wavetables or their sequence, filter cut-off or centre frequency, remaining filter characteristics	Feasible
Sampling	Selection of sound sample	Coarse effect, otherwise requires adaptation of signal modifiers from subtractive method
Indirect methods		
FM	Modulation index, frequency ratio, selection of algorithm	Feasible
Waveshaping	Input signal amplitude, shaping function	Feasible
Physical modelling	Parameters of instrument model and excitation technique	Feasible

5.2. Infeasible Quasi-Physical Systems as Musical Instruments

If physical modelling synthesis can be used to model and listen to various objects, it may be taken a step further. A model is subject to the laws of physics only as much, as its designer decides. Therefore one may discard or alter selected rules or principles to model an imaginary, quasi-physical object that otherwise would be impossible to design and build. Yet such infeasible instrument could still be excited to produce a sound, and might retain enough of real physical features to control it in an intuitive manner.

5.2.1. Concept of Infeasible Instruments

Definition

The term **infeasible instruments** will be used in reference to models of musical instruments, and sound producing objects in general, that cannot be built in real world due to various reasons. Primarily, they might purposely discard or alter chosen laws or principles of physics. They may operate in altered geometry, e.g. they may be hyper-dimensional, exist in a space where some dimensions are looped or warped, or in a space of otherwise altered structure. In a less abstract manner, they may be impossible due to strength of materials, or other material properties. Finally, their geometric or material properties may evolve while producing a sound to the extent impossible for real instruments. Despite any of these properties they need to be able to be modelled, excited to vibrate, and produce sound.

The Reasoning

From the perspective of mechanics or mechanical engineering, modelling of such quasi-physical objects might seem pointless – in principle a model shall describe some aspect of a real world. An object that does not exist cannot provide output data to measure, therefore model of such object has no reference for validation or for directing adjustments of its parameters.

However, from the perspective of sound synthesis such procedure is perfectly rational. The objective of a synthesizer applied in real-time musical performance is to facilitate production and control of varied sounds that have useful musical properties. In this case diverging from existing objects, and in consequence producing possibly different, new sounds, is an advantage. Moreover, a person designing or altering an object that possesses some known properties might use an intuition based on features of real objects as a guideline. A consequence of making some object heavier or larger is predictable. At the same time infeasibility of the instrument introduces some degree of unpredictability to the process. It raises a question 'how would it sound?' and opens the filed for experiments.

5.2.2. Design Outline

By definition an infeasible instrument cannot exist in real world, therefore it is a purely virtual one. It can be implemented as a system consisting of computer programs and physical controllers, requiring a computer to integrate all of its elements (Fig. 5.1). The software layer performs three distinct functions and may be conveniently implemented as separate modules. Firstly, it runs the simulation of the instrument. Secondly, it handles the data incoming from physical controllers. Finally, it manages data received from controllers, translates it to model parameters, and interprets data incoming from a model to produce an output signal.

A modular design allows to distribute the software among several devices in order to avoid compromises between computing capabilities and portability. Model operation shall be carried out using a high performance computing device. Communication

with controller implies its proximity, yet it does not require much computing power, therefore a small, portable device would be adequate. Main part, communicating with remaining two and producing output, needs to be connected to the audio system. All modules can communicate over a computer network, with wireless connection preferred between the main and the controller program.

Depending on the instrument in question, there may be more models operating in parallel, e.g. representing separate vibrating elements, that can be distributed among larger number of computing devices. Similarly, more controllers and controller programs can communicate with the main program. Large, complex models could even be handled by several performers using separate controllers and instances of a controller program.

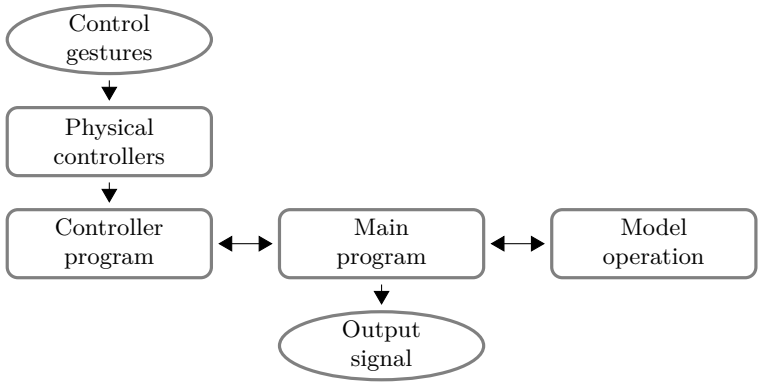


Figure 5.1. Outline of an infeasible instrument implementation design

Consideration of Computing Cost

Quasi-physical models operate using the same techniques and principles as physical models of instruments. In this aspect infeasible instruments shall be regarded as a branch of physical modelling synthesis. One of more straightforward and flexible methods that can be used for their modelling is the finite difference method. In early days of digital sound synthesis such modelling was considered computationally demanding, and in case of real-time operation waveguide synthesis was frequently preferred instead. Later, faster computers allowed to run even relatively complex FD schemes. Still, central processing units (CPUs) of personal computers can meet their limits in case of models operating in more than two dimensions with larger grids, if higher sampling frequency is required.

Large parts of FD schemes may operate in parallel though. Using modular approach the issue of processing power may be solved by carrying out model operation on a device specialised in massively parallel tasks. Among several options three solutions are the most feasible: multi-processor, multi-core systems, field-programmable gate arrays (FPGAs), and graphics processing units (GPUs). The choice of particular solution depends on the form of assumed implementation.

Personal computers utilise multi-core processors, but actual multi-processor systems are a domain of large workstation computers, servers, or server farms. Such systems, particularly server farms, can be scalable and custom-built to meet a specific requirement for processing power. It shall be possible to handle even a complex model by adding sufficient number of computing nodes. It is however the least practical of the three solutions due to size and complexity of required infrastructure. Larger systems have to operate in controlled environment, and can only be accessed remotely, usually through a cloud service, which makes a synthesis process less reliable in real-time performances. On the other hand, performance of smaller multi-processor systems in parallel FD calculations can be matched by far more compact devices.

FPGA is an integrated circuit consisting of programmable logic blocks and reconfigurable interconnections. It can be configured after it has been manufactured, using a hardware description language (HDL), after which it performs similarly to application-specific integrated circuit (ASIC). In comparison to general purpose processors FPGA-based solutions can be faster end more energy efficient. Application of such circuits has been considered for various sound synthesis methods [412, 434] including physical modelling either using waveguides [8] or FD schemes [392, 393]. However, application of FPGAs is limited. FPGA circuit is an additional hardware element which is not a part of standard computer systems. It can only be applied in synthesizers that are complete devices, specified in both, hardware and software layers. It cannot be used in purely software-based implementations, which is currently the most common approach.

The issue of absence of FPGAs in computer systems makes the GPU solution the most feasible of the three. All personal computers are equipped with some form of GPU, either discrete, or integrated with CPU. These devices consist of many parallel processing elements which can be applied to general purpose computing using appropriate programming framework. In case of computations that can be formulated as parallel problems GPUs frequently outperform CPUs. They have been applied in implementations of various synthesis methods, including additive [490] and physical modelling synthesis [530, 531, 254, 440, 63, 144, 442].

5.3. Real-Time FD Simulations Using GPUs

All contemporary computers are equipped with graphics processing units (GPUs). Their original purpose was to take a part of CPU workload related to real-time geometry calculations and image processing while recreating three-dimensional visual environments in video games. Evolving, they gained abilities to perform complex, general purpose operations, but in contrast to CPUs, GPUs were parallel from the very beginning due to specificity of rendering process they were initially aimed at.

Typically a GPU differs from a CPU by having a significantly larger number of simpler cores. This difference can be exploited while designing computer software. Tasks that can be carried out in parallel can be executed on GPUs, while those that have to be performed in a serial manner can be left for less numerous, but faster cores

of CPU. While all PCs and mobile devices are equipped with GPUs, their processing capabilities vary far more than that of CPUs. Basic ones, meant primarily for office computers, have a relatively small number of lower-clocked cores. In contrast, gaming-oriented PCs are equipped with GPUs that are powerful, massively parallel processing units, often characterised by electric power higher than accompanying CPUs.

5.3.1. GPU Programming Framework

GPUs started to be considered an option for general purpose programming in the beginning of the 21st century. One of the first attempts at what has been referred to as GPGPU (General-Purpose GPU), carried out at Stanford University, was a BrookGPU project and a Brook language, with initial reports dating back to 2003 [396, 201]. It had been mainly a research project, and was eventually replaced by two frameworks: CUDA and OpenCL.

Both frameworks can be utilised to program sound synthesizers, though CUDA has been more popular so far [530, 254, 63], possibly due to larger set of available examples, tools, and libraries – either for algebra or signal processing. There are however other important differences between CUDA and OpenCL that need to be considered.

Firstly, CUDA is a proprietary framework, and works with hardware from one GPU vendor only – Nvidia. While this shall lead to better optimised code, CUDA-based software synthesizer would require a PC equipped with Nvidia GPU. On the other hand, OpenCL is an open standard defined by Khronos Group. As such it can be used with hardware from all major vendors, i.e. GPUs from AMD, Intel and Nvidia. Virtually any PC has one of these GPUs, therefore OpenCL code may be considered universally executable.

Secondly, CUDA is an implementation of GPGPU, and allows to program GPUs only. It has the advantage of closer analogies between programming abstractions and elements of GPU, leading to better understanding of processing operation. OpenCL is more general. It aims at heterogeneous computing, i.e. combining processing power of various processing units available, being it a CPU, GPU, DSP, or FPGA. Such universality imposes more abstract nature of programming concepts required to encompass wide variety of processing devices. In effect it is sometimes more difficult to understand a connection between a programming abstraction and an element of GPU. In return OpenCL allows to use all available computing devices simultaneously, which in case of a standard PC allows to combine GPU and CPU, both handled by the same source code.

There have been successful attempts at implementing real-time sound synthesis based on FD schemes using OpenCL framework [440, 144, 442]. Considering its capability to be executed on GPUs from all vendors, scalability through addition of various available computing devices, and openness of the standard, OpenCL will be considered the framework of choice in further deliberations.

5.3.1.1. OpenCL Standard

The specification of OpenCL is divided into four parts, which are referred to as models. They include [396, 201]:

- platform model describing the heterogeneous computer system,
- execution model representing queuing and execution of instructions on a platform,
- memory model defining hierarchy and interactions of OpenCL memory regions,
- programming model mapping a concurrency model onto a physical hardware.

Platform Model

Figure 5.2 presents the OpenCL platform model which describes a heterogeneous system consisting of a host, OpenCL devices, compute units, and processing elements. A platform includes a single host. The host is responsible for I/O operations and can interact with external environment. It also interacts with one or more OpenCL devices, also referred to as compute devices. A compute device is either a CPU, GPU, DSP, FPGA, or other kind of processor supported by the OpenCL. It is divided into compute units, which are subdivided into processing elements.

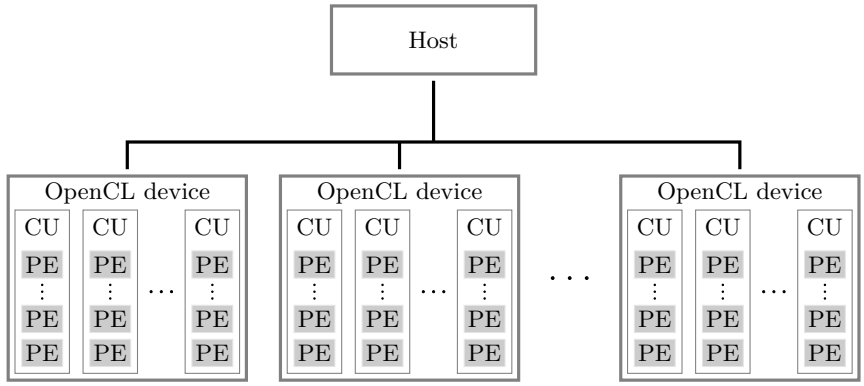


Figure 5.2. An abstract architecture for devices defined by the OpenCL platform model; the platform includes one host device and some number of OpenCL devices; each OpenCL device includes compute units (CU), and each compute unit consists of some number of processing elements (PE)
Source: author's elaboration, based on Munshi et al. [396]

Execution Model

An OpenCL application is executed both on the host and the OpenCL devices. The part executed on host is referred to as the **host program**. The OpenCL devices execute the **kernel**, or a collection of kernels [396]. Kernels are written in the OpenCL C programming language and are compiled by the OpenCL compiler. They usually assume a form of functions operating on memory objects. The execution model defines the way kernels are executed.

A kernel is created on the host and submitted by the host program for execution to the OpenCL device. A one-, two-, or three-dimensional integer index space, referred to as the **NDRange**, is created by the OpenCL runtime, and each point within it executes a single kernel instance. Such instance is referred to as the **work-item** and can be identified using coordinates referred to as global ID. The coordinates are represented by a tuple which size depends on dimensionality of the NDRange. All work-items created by a common host command execute the same sequence of instructions, though the actual work-item behaviour may vary due to branching or different data.

Work-items are grouped into **work-groups** which evenly span the entire index space, i.e. all work-groups are full and have the same size. Work-groups are identified by a group ID. Work-items inside a single work-group are addressed by a local ID. Therefore a work-item is identified either by a global ID or by a local and group ID combination (Fig. 5.3). Only work-items within a common work-group are guaranteed to execute concurrently on processing elements of a single compute unit and share processor resources [396]. Outside of a single work-group the execution of kernels may be serialised.

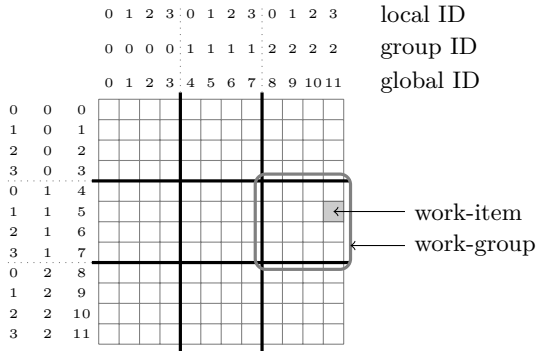


Figure 5.3. NDRange, work-group, and work-item
Source: author's elaboration, based on Munshi et al. [396]

Kernels are defined and executed in the environment referred to as **context**. The context manages OpenCL devices used by the host, kernels, program objects consisting of kernels source code and executables, as well as memory objects which OpenCL devices operate on. While defining a context using an OpenCL API function, the host program queries and chooses a device or its part that will execute kernel functions. Once a device has been chosen the host program can prepare program objects. Source code of kernels is loaded or dynamically generated, and built at runtime. Before kernels can be executed the host program has to define memory objects for use with a particular OpenCL device. Memory objects are moved between the host program and OpenCL devices.

The host interacts with a selected OpenCL device by posting commands to the **command-queue**. There are three command types: kernel execution, memory com-

mands that handle data transfers, and synchronisation commands that allow to control kernels execution order. Queued commands can either execute in order, or out-of-order. In the former case a command can only be started once a previous one has been completed. In the latter case commands start in order, though they do not wait for completion, and any synchronisation has to be explicitly enforced using a mechanism of events. A single context can have multiple associated queues.

Memory Model

Two types of memory objects can be defined in OpenCL: buffer objects and image objects. A buffer can be accessed through pointers, and it is possible to map data structures onto buffers. Image objects are handled by dedicated functions and cannot be accessed directly. OpenCL manages memory consisting of five regions (Fig. 5.4):

- **host memory**, only accessible by the host,
- **global memory**, which can be accessed in read and write mode by work-items from all work-groups, although these operations may be cached, so it is not guaranteed that all work-items observe the same memory state,
- **constant memory**, which is a part of global memory only accessible in read mode,
- **local memory**, restricted to a single work-group only, and shared by all work-items within that group,
- **private memory**, only available to a single work-item.

OpenCL memory model is similar to arrangement utilised by modern GPUs. Even though, it is not a simple one-to-one match. Actual physical representations of abstract memory regions may depend not only on the underlying architecture of the OpenCL device, but also on the program and data complexity. More details regarding regions with an example of matching to a particular GPU can be found in work of Gaster et al. [201].

The host is defined outside of the OpenCL, therefore its memory is separate from memory of the OpenCL devices. If a data needs to be transferred between these regions it requires an explicit copy or map command to be enqueued. Data transfer functions are either blocking or not – the latter return immediately after a command has been enqueued. Local memory values are consistent within a common work-group at work-group synchronisation points. Global memory consistency cannot be guaranteed between groups, although it is consistent for a single work-group at a work-group barrier. Memory consistency between kernels is assured when in-order queue signals completion of all work-items executing a given kernel. In case of out-of-order queue, synchronisation points are required. They are forced by either command-queue barriers, or explicitly through mechanism of events.

Programming Model

A programming model regards mapping of parallel algorithms onto OpenCL. The problem may be approached in two ways: data-parallel or task-parallel. In data-parallel model a single instruction sequence is concurrently applied to elements of

a data structure. An important limitation is the lack of mechanism allowing to synchronise work-items in different work-groups. In a single work-group synchronisation is achieved through the use of work-group barrier – all work items have to encounter a barrier instruction before any of them can process instructions that follow, and at that point they are synchronised.

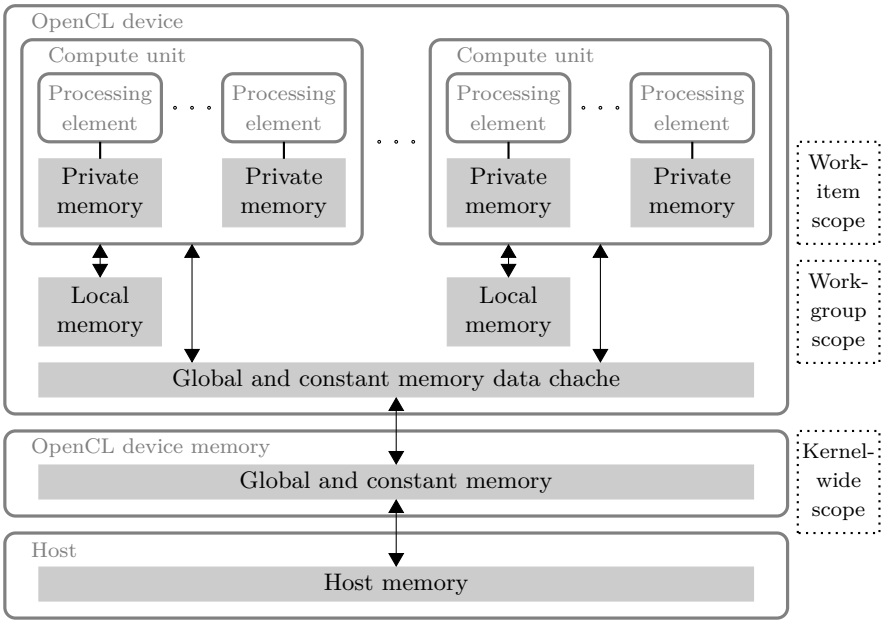


Figure 5.4. OpenCL memory model
Source: author's elaboration, based on Munshi et al. [396] and Gaster et al. [201]

NDRange and its division into work-groups can be defined automatically, which is the implicit model, or by a programmer – in the explicit model of data parallelism. Kernels that have no branch statements execute the same list of instructions, but on a different data. Such model is referred to as the Single Instruction Multiple Data (SIMD). If branches are present, work-items running the same kernel may execute significantly different lists of instruction – it is the Singe Program Multiple Data (SPMD) model. Both models are available in OpenCL. SPMD is more flexible, though in some cases SIMD may be significantly more efficient.

While data parallelism is a default model, task parallelism is available as well. A task is defined as a kernel executed by a single work-item only. Task parallelism can be combined with out-of-order queues and events mechanism to efficiently manage tasks which numbers far exceed number of compute units. Not all OpenCL platforms support such mode though.

5.3.1.2. Heterogeneous Computing

OpenCL goes beyond facilitating general-purpose computations on GPUs by allowing to treat computer systems as heterogeneous platforms, consisting of various computing devices, As a bare minimum, each contemporary computer has a CPU and one or two GPUs[1]. Systems consisting of more CPUs and GPUs are not uncommon though, even as personal computers. In addition, various extension boards or externally connected devices allow to supplement computer systems with DPSs, FPGAs, or other kinds of processing units. Being an open standard, OpenCL can utilise many of these devices.

What is important, by utilising mechanism of platforms, OpenCL allows a single host program to dispatch kernels to various devices. It may be advantageous while designing sound synthesizers based on finite-difference schemes. Separately vibrating elements can be simulated not on one, but on all available computing devices, increasing polyphonic capabilities. In a realistic scenario, one or two CPU cores may handle the host program, user interface and controls, while remaining CPU cores and most of GPU can be benchmarked to determine maximum number of vibrating elements each of them can handle.

It is worth noting, than even without utilisation of GPU OpenCL provides a convenient framework for programming multi-core CPUs in case of data-parallel problems complying to SIMD and SPMD model, including FD schemes. Even in a class of personal computers, contemporary high-end CPUs usually have at least eight, but sometimes as much as thirty two cores. Such systems supplemented with one or two powerful GPUs can handle complex models in real-time, while remaining relatively affordable. Combining it with their reasonable portability and energy efficiency, they are suitable for musical purposes, such as computing facility for a performance-oriented synthesizer. Moreover, they are usually already a part of a recording or an electronic music studio equipment.

5.3.1.3. OpenCL Framework Contents

The OpenCL framework has three components [396]:

- the platform API,
- the runtime API,
- the OpenCL programming language.

A system may have more than one platform available, and each platform may allow to use more than one OpenCL device. Platform API manages platforms, determines available OpenCL devices and their capabilities. Moreover, it defines the context used to run an application. The runtime API handles command queues, manages memory, and compiles OpenCL kernels. Kernels are written in the OpenCL C language, derived from the ISO C99 language.

[1] One basic GPU is usually integrated with CPU, the second, more powerful one, is often added as a separate board for a workstation or video games purpose.

OpenCL Language

The language of kernels is based on C, but a specific subset of C features is not available, i.e. recursive functions, pointers to functions, and bit fields. Standard libraries are limited as well, but particulars depend on the platform and OpenCL version. At the same time, the following set of features has been added:

- vector types and operations, which allows to exploit device parallelism,
- qualifiers to control address spaces related to memory regions,
- built-in functions with functionality necessary for OpenCL programs,
- atomic functions.

When stored as files, OpenCL sources are given 'cl' extension. A very basic example of a kernel written in OpenCL C language is a parallel implementation of addition of two vectors, presented in Listing 5.1.

Listing 5.1. OpenCL kernel for parallel addition of two vectors

```
__kernel void vectorAdd(__global const float *in1, __global const float ↩
    *in2, __global float *out)
{
  int gID = get_global_id(0); // determine address within NDRange space
  out[gID] = in1[gID] + in2[gID]; // perform addition of one element
}
```

Versions

As is the case of all actively developed programming frameworks, OpenCL constantly evolves. For the moment of writing the newest version is 2.2, released in 2017. Each version brings significant enhancements, particularly in productivity domain regarding kernel language. For instance, since version 2.1 the OpenCL C++ kernel language, based on C++14 standard, has been introduced, and version 2.2 expands the set of its object-oriented features. Plans for future releases include converging OpenCL and Vulkan into a single Vulkan API.

However, new features might, or might not be supported by already released processing hardware. Moreover, supported version depends not only on hardware, but also on the combination of operating system, driver, and platform software. For instance, in Majnaro Linux 17.1 with kernel 4.17, AMD-APP software (version 1800.11) supports OpenCL version 2.0, but only for AMD Radeon GPU, while Intel Core i5-6600K CPU is compatible only with version 1.2, released in 2011. The same operating system but with older GPU, namely Nvidia GeForce GT 750M, supports version 1.2 only, while OpenCL ICD loader can support even the newest one.

The bottom line is that new features may be used only if compatibility is not an issue, i.e. a software designed will be executed on tested combination of hardware and system software. Otherwise it is necessary to restrict designed program to one of older, widely supported versions.

5.3.2. Single String

Infeasible instruments share a core of their operation with real-time sound synthesis based on physical modelling of normal instruments. Therefore details of their design and implementation, including parallel computing on GPU, control, and real-time operation, may be discussed using a more standard model. Such an introductory model for FD schemes is typically a string.

5.3.2.1. The Model

An overly simple model with only a few adjustable parameters, such as a wave equation with dissipation (3.127), does not provide much opportunities for expressive control and makes a mediocre performance instrument. A more suitable example would be served by a less basic model of a stiff string with frequency dependent damping, interacting with a hammer. The model may be obtained by starting with a stiff string given by (3.131), and supplementing it with terms containing two loss parameters, as in (3.136). Hammer interaction can be modelled according to (3.160) and (3.161). The resulting model, with variables scaled according to (3.110), assumes the following form

$$\begin{aligned} \frac{\partial^2 u}{\partial t^2} &= \gamma^2 \frac{\partial^2 u}{\partial x^2} - \kappa^2 \frac{\partial^4 u}{\partial x^4} - 2\sigma_0 \frac{\partial u}{\partial t} + 2\sigma_1 \frac{\partial^3 u}{\partial t \partial x^2} + \epsilon(x) \mathcal{M} \tilde{F} \\ \tilde{F} &= -\frac{\mathrm{d}^2 u_H}{\mathrm{d}t^2} = \omega_H^{\alpha+1} \left(\left[u_H - \langle \epsilon, u \rangle_{\mathbb{U}} \right]^+ \right)^\alpha \end{aligned} \tag{5.1}$$

Parameters γ, κ, and $\mathcal{M}$ are defined in (3.111), (3.132), and (3.158), respectively.

5.3.2.2. Finite Difference Scheme

A model given by (5.1) can be approximated using the following explicit differential scheme, which combines schemes (3.139), (3.162), and (3.163)

$$\begin{aligned} \delta_{tt} u_l[n] &= \gamma^2 \delta_{xx} u_l[n] - \kappa^2 \delta_{xxxx} u_l[n] - 2\sigma_0 \delta_{t\cdot} u_l[n] + 2\sigma_1 \delta_{t-} \delta_{xx} u_l[n] + \epsilon \mathcal{M} \tilde{F} \\ \tilde{F} &= -\delta_{tt} u_H[n] = \omega_H^{\alpha+1} \left(\left[u_H[n] - \langle \epsilon, u_l[n] \rangle_{\mathbb{U}_N} \right]^+ \right)^\alpha \end{aligned} \tag{5.2}$$

String part of the scheme expands into

$$\begin{aligned} &\frac{1}{T^2} \left(u_l[n+1] - 2u_l[n] + u_l[n-1] \right) = \\ &= \frac{\gamma^2}{X^2} \left(u_{l+1}[n] - 2u_l[n] + u_{l-1}[n] \right) - \\ &- \frac{\kappa^2}{X^4} \left(u_{l+2}[n] - 4u_{l+1}[n] + 6u_l[n] - 4u_{l-1}[n] + u_{l-2}[n] \right) - \\ &- \frac{\sigma_0}{T} \left(u_l[n+1] - u_l[n-1] \right) + \\ &+ \frac{2\sigma_1}{TX^2} \left(u_{l+1}[n] - 2u_l[n] + u_{l-1}[n] - u_{l+1}[n-1] + 2u_l[n-1] - u_{l-1}[n-1] \right) + \\ &+ \epsilon \mathcal{M} \tilde{F} \end{aligned} \tag{5.3}$$

which results in the following recursion

$$
\begin{aligned}
u_l[n+1] = {} & \frac{2\left(1-\lambda^2-3\kappa^2T^2X^{-4}-2\sigma_1TX^{-2}\right)}{\sigma_0T+1}u_l[n]+ \\
& + \frac{\lambda^2+2\left(2\kappa^2T^2X^{-4}+\sigma_1TX^{-2}\right)}{\sigma_0T+1}\left(u_{l+1}[n]+u_{l-1}[n]\right)+ \\
& + \frac{-\kappa^2T^2X^{-4}}{\sigma_0T+1}\left(u_{l+2}[n]+u_{l-2}[n]\right)+ \\
& + \frac{-1+\sigma_0T+4\sigma_1TX^{-2}}{\sigma_0T+1}u_l[n-1]+ \\
& + \frac{-2\sigma_1TX^{-2}}{\sigma_0T+1}\left(u_{l+1}[n-1]+u_{l-1}[n-1]\right)+\frac{T^2}{\sigma_0T+1}\epsilon\mathcal{M}\tilde{F}
\end{aligned}
\tag{5.4}
$$

A future grid point value is calculated using eight points from two time steps. Values of five points, central as well as nearest and next neighbours, are required from the current step. Three additional points, central and nearest neighbours, are required from the previous one. Stencil of the scheme is presented in Figure 5.5.

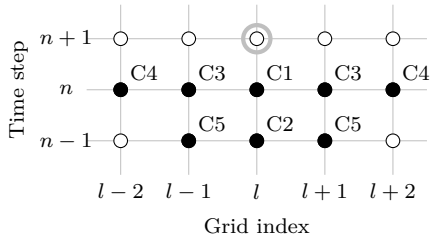


Figure 5.5. A stencil of FD scheme (5.2); black dots symbolise grid points used for approximation, and a gray circle indicates the approximated point; C-symbols indicate grid point coefficients, as used in further presented host program (Listing 5.6)

The minimal value of spatial grid spacing X is related to temporal grid spacing T by the stability condition (3.134), and depends on the stiffness parameter κ. T is either directly equal to sampling frequency or related by a simple ratio, in case the output signal is resampled. Therefore it is unlikely to change its value during synthesis. Stiffness however has easily audible and interesting effect. It might be useful to make it a user-controllable parameter, adjustable during a performance. In effect, X will not only be determined when the model is initiated, but the stability condition will have to be checked after each adjustment of κ to correct grid spacing if required. Grid spacing affects the grid size as well. If a string is supposed to sound during such changes, and string length is not supposed to change, its state has to be preserved by interpolating values from the old grid to the new one.

The hammer is considered as a lumped element, and only later made spatial by applying ϵ distribution. Therefore it is modelled as a simple oscillator with a force term. In each step the force is calculated according to the second part of (5.2), and

its value is used to update the hammer scheme, which may be expanded into

$$\tilde{F} = -\frac{1}{T^2}\left(u_H[n+1] - 2u_H[n] + u_H[n-1]\right) \tag{5.5}$$

leading to the following recursion

$$u_H[n+1] = 2u_H[n] - u_H[n-1] - \tilde{F}T^2 \tag{5.6}$$

The entire modelled system is simulated by iterating three stages, as presented in Figure 5.6. In the first stage the hammer force is determined according to the second line of (5.2), which involves comparison between the last hammer position and the inner product of the ϵ distribution with a string state represented by its entire grid function u in the current time step. Next, the hammer scheme is updated according to (5.6). Finally, in the third step the same force value is utilised to update the string scheme, as given by (5.4). Afterwards, data structures containing string grid and hammer state in next, current, and previous steps are shifted, so that current becomes previous, and next becomes current. This concludes a single step of the simulation, and once all grid points have been updated, the procedure is iterated. A user can interact with the simulation only after the whole step of the iteration has been completed. A typical interaction is an instant change of hammer velocity, which represents a strike attempt.

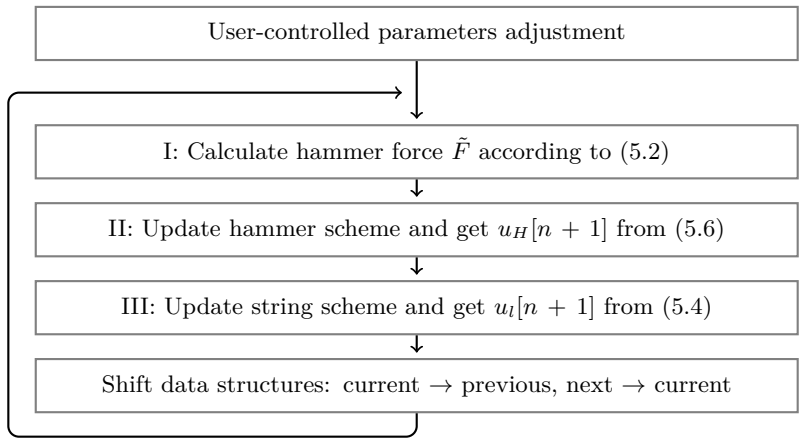


Figure 5.6. A schematic overview of a single step of the simulation for a single grid point l, based on the string and hammer model (5.2); all grid points need to be updated before iterating

5.3.2.3. Implementation Considerations

It would be easy to write a serial implementation of the scheme (5.2), particularly as a non-interactive script in GNU Octave or Matlab language, where ready to use matrix operators are available. However, a parallel, controllable, real-time implementation is significantly more complex, with the following issues to consider:

- division of algorithm into parts that can, and that cannot be executed in parallel,
- synchronisation and access to neighbouring grid points,
- choice of a parallel programming model.

Regarding the first issue, it may be stated that due to grid-based nature of a model, majority of required operations can be executed in parallel. All the grid can be updated simultaneously, with grid points processed concurrently. There are exceptions though. The first one is caused by an inner product while calculating the hammer force. After parallel multiplication, a sum of all elements have to be determined, which will momentarily reduce the number of active tasks. At this moment most part of a processing device will not be utilised. The same applies to updating the hammer scheme. The hammer is simulated by a single, lumped element, and there is no point in calculating the same result in each processing element. The most computationally demanding part, i.e. updating the string scheme, is parallel though.

The second issue involves the scheme stencil. A fully parallel task would not depend on data from other tasks. However, in order to update value of a point belonging to the string grid, it is necessary to use values of neighbouring grid points, which are handled by separate tasks. Therefore, grid values have to be stored in a memory region shared between all tasks updating the same grid. Moreover, all the grid has to be synchronised – at a given moment the same time step n have to be updated in all grid points.

The third issue has to be considered regarding choices of programming models available in OpenCL. Out of two possible, grid-based FD scheme can be implemented as data-parallel SPMD model. SIMD is not a valid option due to branching required to implement parts where parallel algorithm reduces to a single active task, while the remaining tasks wait for it to finish. Parallelism will be determined by the grid size. NDRange will have a single dimension. The number of work-items needs to at least match the size of grid, so that a single work-item handles a single grid point. It may be rounded up to a size preferred by the computing device. Importantly, all work-items have to belong to a single work-group. Thus they can utilise local shared memory without a need for a kernel to exit and be dispatched again in order to perform synchronisation after each time step.

Even though the simplest solution would be to allow a kernel to calculate only a single time step and exit, which would eliminate the requirement for synchronisation during kernel execution, and would allow to simultaneously use more than a single work-group, dispatching a kernel has a significant overhead which slows down the simulation considerably [442]. Therefore, for the sake of efficiency, in order for a kernel to calculate a longer series of time steps in a single run, and thus producing not a single signal sample but a whole buffer, synchronisation within work-items executing the same kernel is required, and hence the requirement for a grid to fit in a single work-group.

5.3.2.4. Program Design

An overview of the program implementation is presented in Figure 5.7. The program has two parts. The first one, executed once, is responsible for various initiali-

sation procedures, e.g. regarding OpenCL or audio engine. It is also here, where the simulation framework is prepared, which includes initialising all parameters of the model and preparing memory representations of modelled objects: string and hammer. The order of operations is relevant due to dependence of later operations on the parameters obtained in preceding ones. For instance, it is only possible to set the grid size and spacing once both, OpenCL device capabilities and simulation parameters are known. Grid size, in turn, is required to create grid arrays, etc.

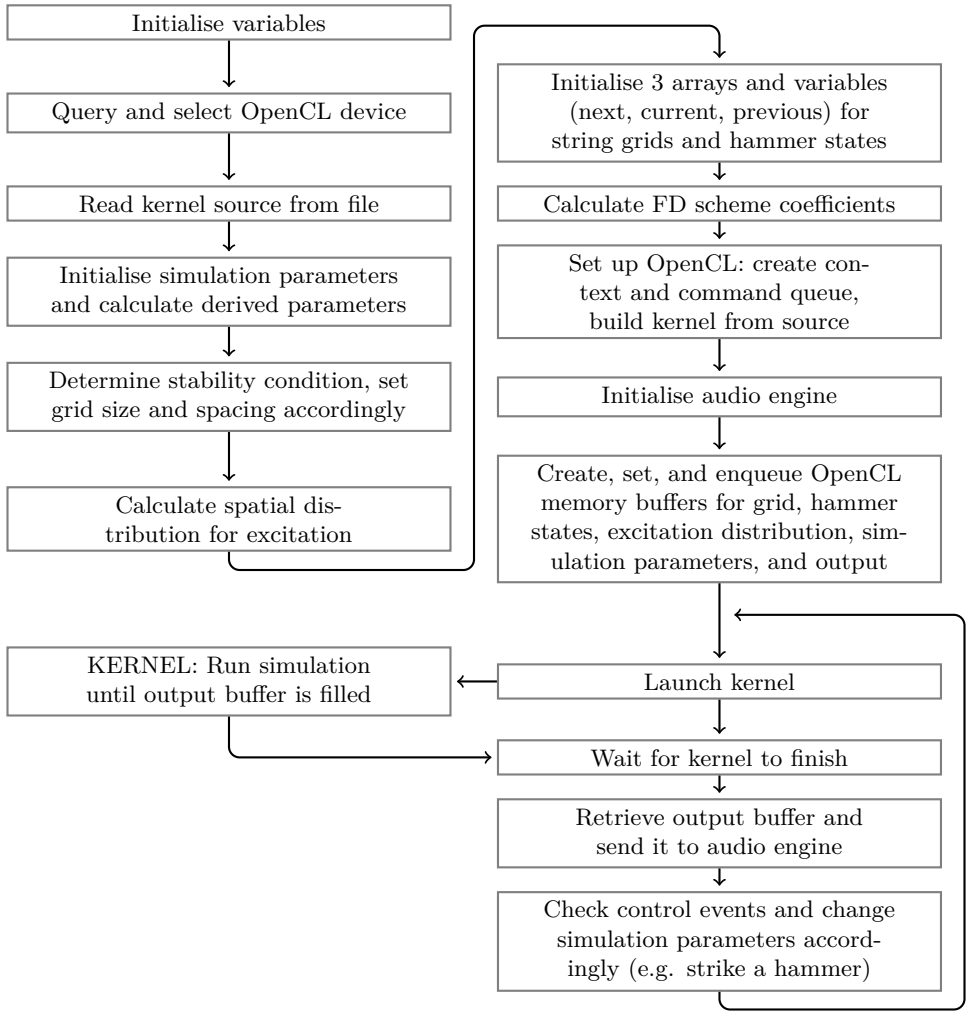


Figure 5.7. A schematic overview of the execution flow diagram for the string and hammer program implemented using OpenCL

The second part of the program contains its main loop. It starts by launching the kernel and waits for it to finish calculations. Kernel runs the actual simulation on the

OpenCL device for a number of steps given by a length of the audio buffer. Each step of a simulation produces a single signal sample, which is obtained by reading value of a grid point specified by readout location. Multi-channel output can be produced by reading values from several points. Once the buffer is full, the host program retrieves it and sends it to the audio device for reproduction. At this point any user-generated control events may be handled in order to change simulation parameters. The most important is activation of a hammer, which instantly resets its position and increases velocity.

Values of grid function u, representing a state of string, are stored in a grid-sized one-dimensional array. In case of a sound synthesizer which can work for a relatively long time, it would be highly inefficient to store grid values for all previous time steps[2]. The scheme (5.4), utilised in simulation, uses two preceding time steps to calculate a new grid point value, therefore it is only necessary to store three steps at a time. The program stores these three grids, 'new' $(n+1)$, 'current' (n), and 'previous' $(n-1)$, in a single, triple-sized array, aligned one after the other. For the sake of efficiency they are utilised in a form of a circular buffer, i.e. the actual data is not moved when the simulation progresses. Only the assignment of array parts is updated (Fig. 5.8). Other data requiring similar updates, e.g. hammer states, is handled in the same way.

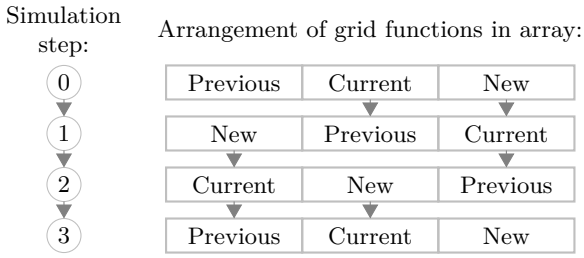


Figure 5.8. Operation of a circular buffer storing string grids representing three consecutive time steps; when simulation progresses, part of an array representing a new grid becomes current, and a current one becomes previous; their data is preserved in the same place, only assignment of arrays changes; data from a grid formerly previous will not be needed further, and in a next step it will be overwritten with data of a new grid

5.3.2.5. User-Controllable Instrument Parameters

One of main strengths of the physical modelling synthesis is control over model parameters that represent features easily recognisable for a performer. A collection of user-controllable parameters describing implemented string and hammer system is presented in Table 5.2.

Change in any of parameters in question requires recalculation of a set of derived parameters. Adjustments of some parameters, e.g. `MR`, may only require to recalculate

[2] 20 minutes of operation with 44100 Hz sample rate would produce approximately 5.3×10^7 signal samples, and equal number of grids. For a grid consisting of 80 points represented by 4-byte `float` type it would result in almost 17 GB of data.

one or more FD scheme coefficients. However, there are also parameters, such as `B`, that once changed, will require to adjust grid spacing and grid size. In order to preserve string state during such adjustments, so that the instrument continues to sound, albeit the sound itself changes, its grid functions will have to be interpolated.

Table 5.2. Synthesis-related simulation parameters that may be controlled by user-generated events in the program simulating string struck with a hammer; initial values provided do not represent any particular real-world instrument, but produce distinctively pitched sound, and constitute a convenient starting point for adjustments

Symbol	Type	Initial value	Parameter
`SR`	`int`	44100	Sample rate [Hz]
`xH0`	`float`	-0.001	Initial hammer-string distance
`vH0`	`float`	200.0	Initial hammer velocity
`MR`	`float`	10.0	Hammer/string mass ratio
`wH`	`float`	15000.0	Hammer stiffness parameter
`alpha`	`float`	2.0	Hammer stiffness non-linearity exponent
`B`	`float`	0.001	String inharmonicity
`f0`	`float`	any	String fundamental frequency [Hz]
`ctr`	`float`	0.1	Location of striking centre (normalised)
`wid`	`float`	0.05	Width of excitation distribution (normalised)
`rp`	`float`	0.3	Readout location (normalised)
`lf1`	`float`	100.0	Loss parameter: low frequency [Hz]
`lT1`	`float`	10.0	Loss parameter: T_{60} for low frequency [s]
`lf2`	`float`	1000.0	Loss parameter: high frequency [Hz]
`lT2`	`float`	8.0	Loss parameter: T_{60} for high frequency [s]

5.3.2.6. Host Program

Parallel implementation of the string and hammer model is based on the OpenCL framework, therefore it consists of the host program, and the kernel. The host program has been written in the C language, with target platform being a Linux PC. A cross-platform PortAudio library has been utilised as an audio engine. User-control has been implemented as a separate program. The host program receives control data over a network, through the User Datagram Protocol (UDP). This mechanism will be discussed in further sections, though it will be omitted here, because it has no relevance for the model implementation itself. Only selected fragments of source code are included and discussed.

One of initial parts of the host program, presented in Listing 5.2, is responsible for determining grid size. Every time any of parameters present in the following fragment is adjusted, a grid size has to be recalculated. This in turn requires an update to all values and data structures that depend on it, such as excitation distribution.

Listing 5.2. Calculation of grid spacing and grid size on the basis of stability condition

```
cl_float X; // spatial grid spacing
int GridSize; // grid size (number of grid points)
int DomainSize; // domain size (with additional boundary points)
X = sqrt(((gamma*gamma*T*T) + sqrt((pow(gamma,4.0)*pow(T,4.0)) + ↩
    (16*K*K*T*T)))/2.0);
GridSize = floor(1.0/X);
X = 1.0/(cl_float)GridSize;
DomainSize = GridSize+2;
```

`MAXGRID` constant, set through `#define` preprocessor directive, represents maximum value of a domain size rounded to the work-group size multiple preferred by the computing device. It is established before compilation, using an assumed borderline case, i.e. a combination of the lowest fundamental frequency and the lowest stiffness parameter allowed to be set. Its primarily role is to allocate and align data arrays. An alternative implementation could be based on dynamically allocated arrays, however due to possibility of frequent grid size changes it would increase recalculation overhead with no significant gains.

Loss parameters σ_0 and σ_1 are calculated according to (3.137), as in Listing 5.3, on the basis of more intuitive time and frequency values.

Listing 5.3. Calculation of loss parameters

```
zeta1 = ((-1.0*gamma*gamma) + sqrt(pow(gamma,4.0) + ↩
    (4.0*K*K*(2.0*M_PI*lf1)*(2.0*M_PI*lf1))))/(2.0*K*K);
zeta2 = ((-1.0*gamma*gamma) + sqrt(pow(gamma,4.0) + ↩
    (4.0*K*K*(2.0*M_PI*lf2)*(2.0*M_PI*lf2))))/(2.0*K*K);
sig0 = 6.0*log(10.0) * ((-1.0*zeta2/lT1)+(zeta1/lT2))/(zeta1-zeta2);
sig1 = 6.0*log(10.0) * ((1.0/lT1)-(1.0/lT2))/(zeta1-zeta2);
```

Spatial distribution of excitation has been approximated with raised cosine function (3.120), as presented in Listing 5.4. It has an adjustable centre position and width. The former parameter represents a striking point, while the latter is related to the size of a hammer head. Other spatial distributions may be used as well, either in form of analytical, or sampled functions. Though if excitation width is to be adjustable, sampled functions would require interpolation.

Listing 5.4. Calculation of excitation distribution – raised cosine

```
cl_float rc[MAXGRID];
for (i=0; i<DomainSize; i++)
{
  dist = sqrt(((((cl_float)i)*X) - ctr) * ((((cl_float)i)*X) - ctr));
  if (dist-(wid/2.0)<0.0) rc[i] = 0.5*(1.0 + cos(2.0*M_PI*dist/wid));
  else rc[i] = 0.0;
}
for (i=DomainSize; i<MAXGRID; i++) rc[i] = 0.0;
```

As stated before, string and hammer states, u and u_H respectively, are stored in triple arrays and variables, for three simulation stages: new, current, and previous (Listing 5.5). Values of array `circ` indicate which element of a triplet represents a particular stage. The string is initialised to the rest position, i.e. its grid function is filled with zeros. The hammer is set to striking state, therefore the synthesizer plays at start. It can be prevented by setting all of `uH` to zeros.

Listing 5.5. Initialisation of storage for string grid and hammer state in three subsequent time steps

```
cl_float uH2; // previous hammer position for initialisation
cl_float uH1; // current hammer position for initialisation
cl_float uH[3]; // hammer states: next, current, previous
cl_float u[MAXGRID*3]; // string states (grids): next, current, previous
cl_float *out; // output buffer
cl_int circ[3]; // circular buffer pointer
circ[0] = 2; // next (n+1) buffer offset multiplier
circ[1] = 1; // current (n) buffer offset multiplier
circ[2] = 0; // previous (n-1) buffer offset multiplier
uH2 = xH0;
uH1 = xH0+(T*vH0);
out = (cl_float *)malloc(bufLen*sizeof(cl_float));
for (i=0; i<bufLen; i++) out[i] = 0.0;
for (i=0; i<(MAXGRID*3); i++) u[i] = 0.0;
uH[circ[0]] = 0.0;
uH[circ[1]] = uH1;
uH[circ[2]] = uH2;
```

Parameters used in the scheme update expression (5.4) are grouped into six scheme coefficients, as presented in Listing 5.6. Five of them (`C1`–`C5`) are further assigned to particular scheme stencil points (Fig. 5.5), and one to a force term (`C6`).

Listing 5.6. Calculation of string scheme coefficients

```
cl_float C1, C2, C3, C4, C5, C6;
C1 = (2.0 - (2.0*gamma*gamma*T*T/(X*X)) - (6.0*K*K*T*T/(X*X*X*X)) - ↩
    (4.0*sig1*T/(X*X))) / (1.0 + (sig0*T));
C2 = (-1.0 + (sig0*T) + (4.0*sig1*T/(X*X))) / (1.0 + (sig0*T));
C3 = ((gamma*gamma*T*T/(X*X)) + (4.0*K*K*T*T/(X*X*X*X)) + ↩
    (2.0*sig1*T/(X*X))) / (1.0 + (sig0*T));
C4 = (-1.0*K*K*T*T/(X*X*X*X)) / (1.0 + (sig0*T));
C5 = (-2.0*sig1*T/(X*X)) / (1.0 + (sig0*T));
C6 = (T*T*MR) / (1.0 + (sig0*T));
```

Single variables that have to be sent to kernel are packed into two parameter arrays, depending on type, `float` or `int` (Listing 5.7).

Listing 5.7. Packing of scheme parameters into arrays for sending to kernel

```
cl_float fpar[10]; // float type parameters
fpar[0] = T; fpar[1] = X; fpar[2] = wH; fpar[3] = alpha;
fpar[4] = C1; fpar[5] = C2; fpar[6] = C3;
fpar[7] = C4; fpar[8] = C5; fpar[9] = C6;
cl_int ipar[6]; // int type parameters
ipar[0] = circ[0]; ipar[1] = circ[1]; ipar[2] = circ[2];
ipar[3] = bufLen; ipar[4] = DomainSize; ipar[5] = rpi;
```

After setting the OpenCL computation environment by calling functions `clCreateContext`, `clCreateCommandQueue`, and `clCreateProgramWithSource`, audio engine is initialised, i.e. it is ready to receive buffers to play. Before the main loop can start, OpenCL memory buffers have to be created, set as kernel arguments, and filled with relevant data, as in Listing 5.8. The buffers include string grids `u`, hammer states `uH`, excitation distribution `rc`, integer and floating point parameters (`ipar` and `fpar`), and output `out`. All except the last one are two-directional, i.e. can be written and read. The output though only needs to be read. Once buffers data have been written, it is stored until release. It allows to run kernel code multiple times, in a loop, without a need to retrieve and resend buffers in each pass. The data stays in the OpenCL device memory between kernel runs, so next kernel continues with a state of the model left by a previous one.

Listing 5.8. Setting memory buffers and kernel arguments

```
cl_mem bufferU = clCreateBuffer(context, CL_MEM_READ_WRITE, ↩
    3*MAXGRID*sizeof(cl_float), NULL, &error);
cl_mem bufferUH = clCreateBuffer(context, CL_MEM_READ_WRITE, ↩
    3*sizeof(cl_float), NULL, &error);
cl_mem bufferE = clCreateBuffer(context, CL_MEM_READ_WRITE, ↩
    MAXGRID*sizeof(cl_float), NULL, &error);
cl_mem bufferI = clCreateBuffer(context, CL_MEM_READ_WRITE, ↩
    6*sizeof(cl_int), NULL, &error);
cl_mem bufferF = clCreateBuffer(context, CL_MEM_READ_WRITE, ↩
    10*sizeof(cl_float), NULL, &error);
cl_mem bufferO = clCreateBuffer(context, CL_MEM_READ_WRITE, ↩
    bufLen*sizeof(cl_float), NULL, &error);

clSetKernelArg(kernel, 0, sizeof(cl_mem), (void*)&bufferU);
clSetKernelArg(kernel, 1, sizeof(cl_mem), (void*)&bufferUH);
clSetKernelArg(kernel, 2, sizeof(cl_mem), (void*)&bufferE);
clSetKernelArg(kernel, 3, sizeof(cl_mem), (void*)&bufferI);
clSetKernelArg(kernel, 4, sizeof(cl_mem), (void*)&bufferF);
clSetKernelArg(kernel, 5, sizeof(cl_mem), (void*)&bufferO);
```

```
if (clEnqueueWriteBuffer(commandQueue, bufferU, CL_TRUE, 0, ↩
    3*MAXGRID*sizeof(cl_float), u, 0, NULL, NULL)!=CL_SUCCESS) ↩
    printf("Write buffer error: u\n");
if (clEnqueueWriteBuffer(commandQueue, bufferUH, CL_TRUE, 0, ↩
    3*sizeof(cl_float), uH, 0, NULL, NULL)!=CL_SUCCESS) printf("Write ↩
    buffer error: uH\n");
if (clEnqueueWriteBuffer(commandQueue, bufferE, CL_TRUE, 0, ↩
    MAXGRID*sizeof(cl_float), rc, 0, NULL, NULL)!=CL_SUCCESS) ↩
    printf("Write buffer error: rc\n");
if (clEnqueueWriteBuffer(commandQueue, bufferI, CL_TRUE, 0, ↩
    6*sizeof(cl_int), ipar, 0, NULL, NULL)!=CL_SUCCESS) printf("Write ↩
    buffer error: ipar\n");
if (clEnqueueWriteBuffer(commandQueue, bufferF, CL_TRUE, 0, ↩
    10*sizeof(cl_float), fpar, 0, NULL, NULL)!=CL_SUCCESS) printf("Write ↩
    buffer error: fpar\n");
```

The main loop of the host program contains a kernel enqueue function, which creates one-dimensional NDRange address space and runs a kernel in each of its work-items. Size of both address spaces, local (`locWGS`) and global (`globWGS`), is set to `MAXGRID`, so only one work-group is utilised. This allows work-items to share their local memory and perform synchronisation on barriers. The second function enqueues buffer read operation, which attempts to retrieve output signal from kernel in a blocking mode, set by the third argument (`CL_TRUE` value). A blocking read returns only when the data is ready. Due to default 'in order' queue execution, the read will start after the kernel has finished. Therefore the host program waits here for the OpenCL device to finish the kernel.

Listing 5.9. Launching kernels and retrieving output

```
if (clEnqueueNDRangeKernel(commandQueue, kernel, 1, NULL, globWGS, locWGS, ↩
    0, NULL, NULL)!=CL_SUCCESS) printf("Enqueue kernel error\n");
if (clEnqueueReadBuffer(commandQueue, bufferO, CL_TRUE, 0, ↩
    bufLen*sizeof(cl_float), out, 0, NULL, NULL)!=CL_SUCCESS) ↩
    printf("Output retrieve error\n");
```

Once the output has been read into array `out`, the host program waits for the audio engine to be ready to receive next buffer to play. When it is ready, `out` is sent for playback, and the program continues. At this point host program receives and interprets control data from a network connection. Depending on parameter which needs to be adjusted, relevant data may be required to be read from, modified, and written to the OpenCL device memory buffer with functions `clEnqueueReadBuffer` and `clEnqueueWriteBuffer`. This concludes a single pass of the main loop.

The end-of-loop condition is triggered either by receiving 'exit' command from the user, or by reaching previously set maximal simulation time. In the end, memory objects are released using `clReleaseMemObject` function. All required audio-, network-, and OpenCL-related termination procedures are carried out, dynamically allocated memory is freed, and the program ends.

Presented implementation, based on a standard main loop, is aimed towards operation in modularised synthesis system, where it would run as a 'headless' engine, or back-end. Control and user interaction are tasks of other modules, communicating with the synthesis engine over a network. If all modules of the synthesizer were to be integrated into a monolithic, interactive program, main loop design would be replaced by a system of callback functions and events.

5.3.2.7. Kernel

The actual operation of a model is performed by the OpenCL kernel. The kernel is enqueued to run on a number of work-items at least equal to the domain size, which is a grid size increased by boundary points. Not all operations however, can be performed in parallel, which is indicated in Figure 5.9. During these operations unused work-items have to wait for the result, which is implemented using `barrier` function which only returns, once all work-items have reached it.

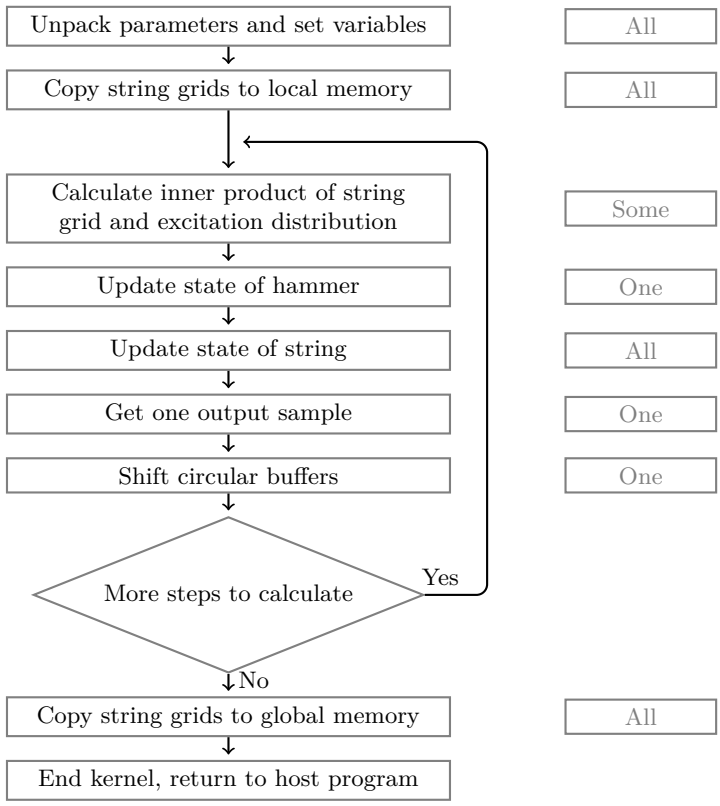


Figure 5.9. A schematic overview of the execution flow diagram for the OpenCL kernel performing simulation of string and hammer; right column indicates work-item parallelism, i.e. the number of work-items simultaneously performing given operation

The kernel header (Listing 5.10) lists all of its arguments, which are consistent with a fragment of the host program that assigns them with memory buffers (Listing 5.8). All arguments are pointers to arrays located in a global memory region.

Listing 5.10. Kernel header

```
__kernel void StringHammerSim(__global float* gu, __global float* uH, ↩
    __global float* exc, __global int* ipar, __global float* fpar, ↩
    __global float *out)
```

A work-item determines its address in the NDRange space using functions `get_global_id` and `get_local_id`. In the string simulation the space is one-dimensional. In the host program `MAXGRID` has been assumed equal to 256, and it has been set as a fixed value in kernel. Majority of parameters packed by the host program into `ipar` and `fpar` arrays are unpacked into private variables, separate for each work-item. Remaining variables, i.e. pointers of a circular buffer `c1`–`c3`, local copy of the string grid `u[]`, hammer force `fH`, and `tprod` array used to calculate inner product, have a `local` address space qualifier, which makes them shared between all work-items.

Listing 5.11. Unpacking received simulation parameters and setting variables

```
int n, j, gid = get_global_id(0), lid = get_local_id(0);
local int c0;
local int c1;
local int c2;
int DomainSize = ipar[4];
int rpi = ipar[5];
int len = ipar[3];
local float u[256*3];
local float tprod[256];
local float fH;
float ip, t1, t2, t3, t4;
float T = fpar[0];
float X = fpar[1];
float wH = fpar[2];
float alpha = fpar[3];
float P1 = fpar[4];
float P2 = fpar[5];
float P3 = fpar[6];
float P4 = fpar[7];
float P5 = fpar[8];
float P6 = fpar[9];
```

In the beginning (Listing 5.12), the first work-item determines the state of circular buffers left by previous run of the kernel. The remaining work-items wait on a barrier. Afterwards, string grid state is copied from the global array `gu` to the local one. The operation is parallel – each work-item copies only one array cell.

Listing 5.12. Moving data to local memory

```
if (lid==0)
{
   c0 = ipar[0]*DomainSize;
   c1 = ipar[1]*DomainSize;
   c2 = ipar[2]*DomainSize;
}
barrier(CLK_LOCAL_MEM_FENCE|CLK_GLOBAL_MEM_FENCE);
u[lid+c0] = gu[lid+c0];
u[lid+c1] = gu[lid+c1];
u[lid+c2] = gu[lid+c2];
barrier(CLK_LOCAL_MEM_FENCE|CLK_GLOBAL_MEM_FENCE);
```

In order to determine the hammer force, the kernel needs to calculate an inner product (Listing 5.13). Its first phase, a cell-by-cell multiplication of two arrays and a scalar `X`, is parallel, and the result is stored in `tprod` array. The second phase is a calculation of a total sum of all array elements. It is performed in parallel, although only part of work items can be utilised, organised in a proper binary tree, as shown in Figure 5.10. Compared to serialised sum, such parallel algorithm reduces a number of summation steps from $(N-1)$ to $\log_2 N$, which in this particular case ($N = 256$) translates to reduction by almost 97%.

Listing 5.13. Calculating inner product

```
if (lid<DomainSize) tprod[lid] = u[lid+c1]*exc[lid]*X;
else tprod[lid] = 0.0;
barrier(CLK_LOCAL_MEM_FENCE);
for (j=128; j>0; j/=2)
{
   if (lid<j) tprod[lid] += tprod[lid+j];
   barrier(CLK_LOCAL_MEM_FENCE);
}
```

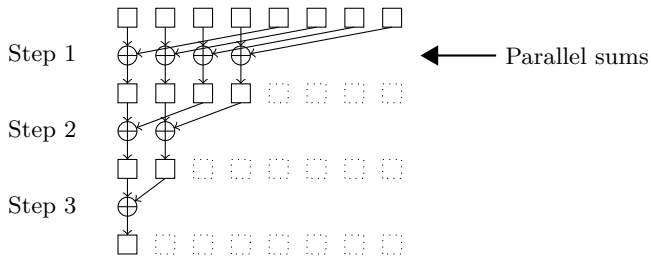


Figure 5.10. Parallel calculation of total sum of array values

In Listing 5.14 inner product is used to calculate the hammer force `fH`, and to update hammer state `uH` using its FD scheme (5.6). There is one hammer and it requires a few operations only, therefore it is handled by a single work-item, while remaining ones have to wait on a barrier.

Listing 5.14. Updating state of the hammer

```
if (lid==0)
{
   ip = tprod[0];
   t1 = wH;
   t2 = 1.0 + alpha;
   t3 = uH[(c1/DomainSize)] - ip;
   t4 = alpha;
   if (t3>0) fH = pow(t1,t2)*pow(t3,t4);
   else fH = 0.0;
   uH[(c0/DomainSize)] = (2.0*uH[(c1/DomainSize)]) - uH[(c2/DomainSize)] - ↩
       (T*T*fH);
}
barrier(CLK_LOCAL_MEM_FENCE);
```

After the force has been calculated, all is ready to update a state of the string, which is carried out by the code presented in Listing 5.15. In majority of cases size of NDRange space will exceed grid size with additional boundary points. An update is performed on the active part of the string, according to the string update expression (5.4), with scheme coefficients calculated earlier in host program. After update, a work-item corresponding to a grid element indicated by readout location stores momentary value of a grid function in this point as a single signal sample.

Listing 5.15. Updating state of the string

```
if ((lid>1)&&(lid<(DomainSize-2)))
{
   u[c0+lid] = (P1*u[c1+lid]) + (P2*u[c2+lid]) + ↩
       (P3*(u[c1+lid-1]+u[c1+lid+1]))
           + (P4*(u[c1+lid-2]+u[c1+lid+2])) + ↩
               (P5*(u[c2+lid-1]+u[c2+lid+1])) + (P6*exc[lid]*fH);
}
barrier(CLK_LOCAL_MEM_FENCE);
if (lid==rpi) out[n] = u[c0+rpi];
barrier(CLK_LOCAL_MEM_FENCE);
```

A single step of the simulation has been completed, but before continuing the the next iteration, circular buffers have to be shifted, which is presented in Listing 5.16. A single shift is carried out by a single work-item, with the rest waiting on a barrier. Once buffer indicators have been shifted, the algorithm returns to calculating inner product, with updated string and hammer states.

Listing 5.16. Shifting circular buffers

```
if (lid==0)
{
   c0 = (c0+DomainSize)%(3*DomainSize);
   c1 = (c1+DomainSize)%(3*DomainSize);
   c2 = (c2+DomainSize)%(3*DomainSize);
}
barrier(CLK_LOCAL_MEM_FENCE);
```

If the output buffer has been filled, the kernel needs to stop, so that data obtained can be retrieved by the host program and sent to audio device to play. Befor exiting, a reverse of kernel initial operations is performed, i.e. circular buffer as well as string state are stored in global memory region (Listing 5.17), so that they will be accessible in the next kernel run.

Listing 5.17. Moving data to global memory

```
if (lid==0)
{
   ipar[0] = c0/DomainSize;
   ipar[1] = c1/DomainSize;
   ipar[2] = c2/DomainSize;
}
barrier(CLK_LOCAL_MEM_FENCE|CLK_GLOBAL_MEM_FENCE);
gu[lid+c0] = u[lid+c0];
gu[lid+c1] = u[lid+c1];
gu[lid+c2] = u[lid+c2];
barrier(CLK_LOCAL_MEM_FENCE|CLK_GLOBAL_MEM_FENCE);
```

Performance of the simulation can be controlled by adjusting size of the output buffer, i.e. number of time steps to calculate in each kernel run. The shorter the buffer, the more fine-grained instrument control is achieved in the time domain, but performance drops due to frequent dispatching the kernel and transferring data from the OpenCL device. Large output buffers though, can lead to audible jumps in case of parameters that are adjusted gradually during sound production, e.g. by sliders on the controller.

5.3.3. Multiple Strings

If a simulation is carried out on a GPU, a single string will utilise only a small part of its processing elements. A logical utilisation of the remaining ones would be to simultaneously simulate larger set of strings, which is often the case of acoustic instruments, such as the grand piano with approximately 230 strings typically struck with 88 hammers.

The instrument implemented will not be a model of piano, but a more abstract, multi-string instrument, with each separate string being struck with an individual hammer. Each string will have a separate set of parameters, mirroring features of pre-

viously implemented single string (Tab. 5.2), therefore each string would not only be able to be tuned differently, but could also have distinct timbre-related characteristics.

Such instrument does not require modifications of the underlying FD model – it may be considered a set of separate instruments, each one being a previously implemented string with a hammer. They will be, however, updated simultaneously, in order to mix their separate outputs into a single, common one.

5.3.3.1. Implementation Considerations

In case of a single string it was necessary to keep all of the string grid in a single work-group. While simulating multiple strings that do not interact with each other, every string will have a separate work-group assigned. Thus one string will share a set of variables stored in local memory, e.g. a grid function, but these variables will have separate instances for different strings. None of the algorithms applied needs to be altered. Only data structures they operate on will be addressed in a more complex manner.

5.3.3.2. Changes in Program Design

In order to accommodate a multiplied number of variables that needs to be grouped on a string basis, the program has to switch from a collection of variables representing a string, to a collection of arrays, representing a string set. Array index will identify particular string, so that string-hammer system `s` will be described by parameters `xH0[s]`, `vH0[s]`, `MR[s]`, etc.

There will be still one, common output buffer sent to the audio engine to play, but it will require mixing to prepare. Each string will produce its own output buffer. Kernel will align these buffers into a single, one-dimensional array, where they will be selectable by adding an offset – a multiple of output length – to the address. The array will be retrieved by the host program, which will mix it (Fig. 5.11).

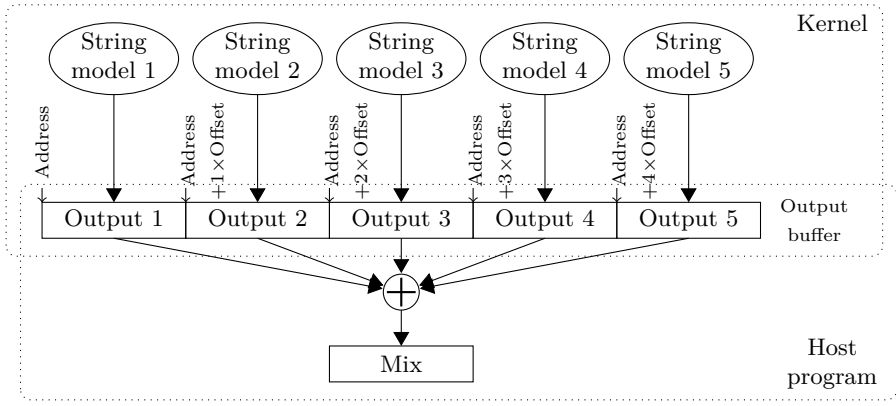


Figure 5.11. Mixing outputs of individual strings

It would be possible to mix outputs in kernel, by an OpenCL device, but host-side leaves more flexibility. It is easy to define various mixing scenarios and produce e.g. different channel arrangements, with adjustable amplifications and delays applied to signals of separate strings.

5.3.3.3. Control Considerations

With multiple strings, each having relatively large set of parameters, control of the instrument may be an issue. Therefore, even though the engine allows to control everything separately, the implementation of control software may limit this functionality in favour of ergonomics. The most basic approach would be to differentiate only a few selected parameters, such as fundamental frequency and damping, while leaving the rest with a collective control, affecting all strings in the same way.

A much better, flexible approach should allow to switch between different control mechanisms. In the first one, collective, a parameter adjustment would affect all strings. The second, enabled on demand, would allow to control every feature of each string separately, either through elaborate computer user interface, or by a large, advanced controller. The third mechanism would allow to define functions, either through expression, by drawing, or sampling, that would vary a parameter among a string set in orderly manner.

An example of the last mechanism is tuning of strings to 12-TET system

$$f_0[i] = 2^{\frac{i+d}{12}} f_{\text{ref}} \tag{5.7}$$

where i is the string index, f_{ref} is the fundamental frequency of a reference pitch, e.g. 442 Hz, and d is the interval in semitones from the reference pitch to the string with index $i = 0$. Value of d is not limited to natural numbers – it may be represented by any reasonably sized real number. Similar relations can be easily defined for damping or hammer parameters.

5.3.3.4. Host Program

A general design of the host program does not vary significantly from the one string case. Yet, there are differences in implementation of multiple strings which need to be further discussed. Initially, the host program defines the number of simulated strings. As shown in Listing 5.18, a preprocessor directive is utilised to facilitate creation of adequately sized arrays of parameters for a string set.

Listing 5.18. Setting the number of strings

```
#define NSTR 12
```

Almost all model parameters are represented by arrays, as shown in the first part of the Listing 5.19. Only a few, such as duration of simulation, or sampling rate, which determines temporal grid interval, are common for all strings. During initialisation some of parameter arrays are set to common values, e.g. `xH0`, while others, like `f0`, are varied according to chosen expression.

Listing 5.19. Setting parameters of a string set

```
int nstr = NSTR; // number of strings - a variable
cl_float xH0[NSTR]; // array: separate value for each string
// ...
cl_float dur = 300; // variable: common value for all strings

for (i=0; i<nstr; i++)
{
   xH0[i] = -0.001;
   // ...
}
f0[0] = 110.5;
for (i=1; i<nstr; i++) f0[i] = f0[i-1]*pow(2.0,(1.0/12.0));
```

Different string parameters imply that each string may have a different grid size. It needs to be calculated separately, which is shown in Listing 5.20, where the only parameter common to all strings is the sampling interval `T`. Remaining parameters vary between strings and are stored in arrays.

Listing 5.20. Calculation of grid spacings for a set of strings

```
cl_float X[NSTR]; // spatial grid spacing
int GridSize[NSTR]; // grid size (number of grid points)
int DomainSize[NSTR]; // domain size (additional boundary points)
for (i=0; i<nstr; i++)
{
  X[i] = sqrt(((gamma[i]*gamma[i]*T*T) + ↩
      sqrt((pow(gamma[i],4.0)*pow(T,4.0)) + (16*K[i]*K[i]*T*T)))/2.0);
  GridSize[i] = floor(1.0/X[i]);
  X[i] = 1.0/GridSize[i];
  DomainSize[i] = GridSize[i]+2;
}
```

The excitation distribution (Listing 5.21) is again of the raised cosine shape. However, due to different grids, as well as possibly different widths and centre positions, it is calculated and stored separately for each string. Distributions for particular strings are aligned in a single one-dimensional array, and padded to `MAXGRID`. Such data structure does not have to be reallocated in case of alterations to grid sizes.

Listing 5.21. Calculation of excitation distributions for a set of strings

```
cl_float rc[MAXGRID*NSTR];
for (j=0; j<nstr; j++)
{
   for (i=0; i<DomainSize[j]; i++)
   {
      dist = sqrt(((((cl_float)i)*X[j])-ctr[j]) * ↩
          ((((cl_float)i)*X[j])-ctr[j]));
```

```
      if (dist-(wid[j]/2.0)<0.0) rc[i+(MAXGRID*j)] = ↩
          0.5*(1.0+cos(2.0*M_PI*dist/wid[j]));
      else rc[i] = 0.0;
   }
   for (i=DomainSize[j]; i<MAXGRID; i++) rc[i+(MAXGRID*j)] = 0.0;
}
```

An arrangement similar to the storage of excitation distribution is applied to store string grids. It is a one-dimensional array, where grids of subsequent strings are aligned and padded to `MAXGRID`. Output buffer `out` is arranged as in Figure 5.11. States of hammers are stored in arrays as well. Initial values are set in loops (Listing 5.22).

Listing 5.22. Initialisation of storage for multiple string grids and states of hammers

```
cl_float uH2[NSTR], uH1[NSTR], uH[3*NSTR];
cl_float u[MAXGRID*3*NSTR]; // string states (grids): next, current, previous
cl_float *out; // output buffer
cl_int circ[3]; // circular buffer pointer
circ[0] = 2; // next (n+1) buffer offset multiplier
circ[1] = 1; // current (n) buffer offset multiplier
circ[2] = 0; // previous (n-1) buffer offset multiplier
for (i=0; i<nstr; i++)
{
   uH2[i] = xH0[i];
   uH1[i] = xH0[i]+(T*vH0[i]);
}
out = (cl_float *)malloc(nstr*bufLen*sizeof(cl_float));
for (i=0; i<(nstr*bufLen); i++) out[i] = 0.0;
for (i=0; i<(MAXGRID*3*nstr); i++) u[i] = 0.0;
for (i=0; i<nstr; i++)
{
   uH[circ[0]+(i*3)] = 0.0;
   uH[circ[1]+(i*3)] = uH1[i];
   uH[circ[2]+(i*3)] = uH2[i];
}
```

Although the FD scheme remains unchanged in comparison to the single string case, a separate set of scheme coefficients is required for each string. Therefore they are stored in arrays `C1[NSTR]`–`C6[NSTR]` and are initialised in a loop using parameters from arrays as well. Considerably increased number of parameters requires a modification in the way they are packed for sending to kernel, which is presented in Listing 5.23. The size of packed structure depends on the number of strings.

Listing 5.23. Packing of multiple scheme parameters into array for sending to kernel

```
cl_float fpar[1+(NSTR*9)]; // float parameters
fpar[0] = T;
for (i=0; i<nstr; i++)
{
```

```
   fpar[1+(i*9)] = X[i];
   fpar[2+(i*9)] = wH[i];
   fpar[3+(i*9)] = alpha[i];
   fpar[4+(i*9)] = C1[i];
   // ...
   fpar[9+(i*9)] = C6[i];
}
cl_int ipar[4+(NSTR*2)]; // int parameters
ipar[0] = circ[0];
ipar[1] = circ[1];
ipar[2] = circ[2];
ipar[3] = bufLen;
for (i=0; i<nstr; i++)
{
   ipar[4+(i*2)] = DomainSize[i];
   ipar[5+(i*2)] = rpi[i];
}
```

An important change applies to the arrangement of the NDRange space, which in case of multiple strings is two-dimensional. The first dimension is related to the grid size as previously. The second one represents subsequent strings (Listing 5.24).

Listing 5.24. Setting size of the NDRange space

```
locWGS[0] = MAXGRID; // local: MAXGRID x 1
locWGS[1] = 1;
globWGS[0] = MAXGRID; // global: MAXGRID x nstr
globWGS[1] = nstr;
```

Due to arrangement of data in one-dimensional arrays, preparation of memory buffers, assigning kernel arguments, and filling buffers with data has not been altered significantly. Only the size of buffers has changed, due to dependence on the number of strings, which is shown in Listing 5.25.

Listing 5.25. Setting memory buffers

```
cl_mem bufferU = clCreateBuffer(context, CL_MEM_READ_WRITE, ↩
    nstr*3*MAXGRID*sizeof(cl_float), NULL, &error);
cl_mem bufferUH = clCreateBuffer(context, CL_MEM_READ_WRITE, ↩
    nstr*3*sizeof(cl_float), NULL, &error);
cl_mem bufferE = clCreateBuffer(context, CL_MEM_READ_WRITE, ↩
    nstr*MAXGRID*sizeof(cl_float), NULL, &error);
cl_mem bufferI = clCreateBuffer(context, CL_MEM_READ_WRITE, ↩
    (4+(nstr*2))*sizeof(cl_int), NULL, &error);
cl_mem bufferF = clCreateBuffer(context, CL_MEM_READ_WRITE, ↩
    (1+(nstr*9))*sizeof(cl_float), NULL, &error);
cl_mem bufferO = clCreateBuffer(context, CL_MEM_READ_WRITE, ↩
    nstr*bufLen*sizeof(cl_float), NULL, &error);
```

In the main program loop the kernel is enqueued similarly (Listing 5.26), only now the third parameter reflects two-dimensional structure of the NDRange space, accommodating all strings. The last instruction mixes outputs of all simulated strings.

Listing 5.26. Launching kernels, retrieving output, and mixing string signals

```
if (clEnqueueNDRangeKernel(commandQueue, kernel, 2, NULL, globWGS, locWGS, ↩
    0, NULL, NULL)!=CL_SUCCESS) printf("Enqueue kernel error\n");
if (clEnqueueReadBuffer(commandQueue, buffer0, CL_TRUE, 0, ↩
    nstr*bufLen*sizeof(cl_float), out, 0, NULL, NULL)!=CL_SUCCESS) ↩
    printf("Output retrieve error\n");
for (i=0; i<bufLen; i++) for (j=1; j<nstr; j++) out[i] += out[i+(j*bufLen)];
```

5.3.3.5. Kernel

All the data sent between the host program and the kernel is kept in one-dimensional arrays, where data regarding subsequent strings is aligned in sequence, and padded if necessary. Therefore kernel header does not differ from the one string variant. What differs, is the length of buffers and higher complexity of their inner organisation.

A work-item checks three components of its address: one-dimensional local ID and global ID along two dimensions. Address along the second dimension of global ID is identified as a string ID, while the first dimension gives an index of a grid point, as shown in Listing 5.27. Work-items sharing a local memory, i.e. belonging to the same work-group, represent elements of the same string. Therefore local variables and arrays, particularly local grid representation `u`, are defined as in the one string case. Private variables representing simulation parameters are unpacked using string ID, e.g. `X=fpar[1+(sid*9)];`, according to packing scheme shown in Listing 5.23.

Listing 5.27. Checking the address of a work-item in two-dimensional NDRange

```
int gid = get_global_id(0), lid = get_local_id(0); // grid point ID
int sid = get_global_id(1); // string ID
```

State of the circular buffers is determined as in the one string case, although with multiple strings each string has its own copy of circular buffer indicator. Progress of simulation is not synchronised between strings, therefore each copy of indicator is shifted independently. By the end of the kernel run though, the same number of steps will have been completed for each string, and the state of all copies will by the same. Thus it is enough to save only the first copy into a global buffer before returning to the host program. String grid data is copied from a global buffer to a local memory with regards to string ID, as shown in Listing 5.28.

Listing 5.28. Moving grid data to local memory

```
u[lid+c0] = gu[lid+c0+(sid*3*256)];
u[lid+c1] = gu[lid+c1+(sid*3*256)];
u[lid+c2] = gu[lid+c2+(sid*3*256)];
```

Calculation of the inner product does not change, although addressing elements of excitation distribution has to include string ID, i.e. `exc[lid+(sid*256)]`. A workgroup shares a single string and a single hammer, therefore the hammer force is calculated as in the one string case. Only the update of the hammer state, which is stored in a global buffer, is altered (Listing 5.29).

Listing 5.29. Updating state of a single hammer

```
uH[(c0/DomainSize)+(sid*3)] = (2.0*uH[(c1/DomainSize)+(sid*3)]) - ↩
    uH[(c2/DomainSize)+(sid*3)] - (T*T*fH);
```

A part of kernel that updates a string uses local variables, thus it does not change as well, with the exception of the aforementioned excitation distribution addressing. Each updated string produces its own output, which is stored with an offset, as shown in Listing 5.30.

Listing 5.30. Storing outputs of all strings

```
if (lid==rpi) out[n+(len*sid)] = u[c0+rpi];
```

Before progressing to the next time step, circular buffer indicators are shifted as in the one string case, and kernel iterates to the next time step. Once number of steps sufficient to fill the output buffer has been calculated, the loop is ended. Circular buffer indicator and state of all strings are copied to the global memory buffer, reversing the initial kernel operations.

5.3.4. Real-Time Control

Presented string model implementation, as well as implementations of models of performance-oriented infeasible instruments, handle user control through an external control module, as proposed in Figure 5.1.

5.3.4.1. Control Procedure Design

The design of the control procedure is presented in Figure 5.12. A user interacts with a physical controller connected to a computer running a controller program. The program connects over the network with a UDP server being a part of the host program. After directing an output buffer for playback to the audio engine, the host program checks for particular control sequences within incoming UDP data. Such sequences are interpreted to obtain new values for given simulation parameter. Necessary data is retrieved from the memory buffers, changes are applied, and adjusted data is sent back to the buffers.

5.3.4.2. Controller Program Implementation

The controller program has been implemented using PureData (Pd) – an open source, cross-platform visual programming language aimed at interactive computer music. Pd is particularly well-suited for tasks involving various music-related control

mechanisms [534]. It handles not only MIDI controllers, but also human interface devices (HIDs) such as gamepads and joysticks, tablets, various sensors, audio input, and even video input, which can be internally analysed to provide control parameters (Fig. 5.12), e.g. to design an *ad hoc* theremin-like controller. The list is not closed. Support for new devices may be relatively easily added due to the open source license, rich documentation, and help of a large community of users and developers.

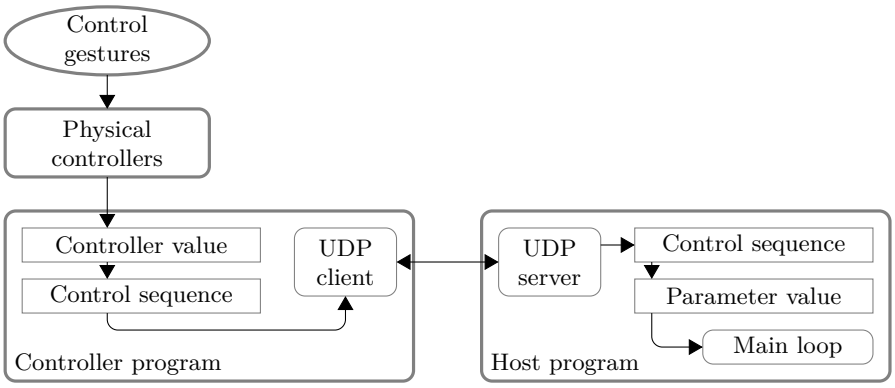


Figure 5.12. Control procedure for the multi-string synthesizer

Four reasons for choosing Pd as an environment of the controller program are:

- wide variety of physical controllers handled, including MIDI and external sensors, which allows to design intuitive control schemes for infeasible instruments,
- network capabilities, facilitating flexible communication with the OpenCL host program running the synthesis engine in the modularised environment,
- cross-platform implementation, allowing to run the controller program on a wide selection of devices, including mobile,
- open source license, allowing to freely modify and extend it, as well as integrate it into a heterogeneous synthesis system.

Pd programs are referred to as 'patches', where objects either processing signals, or performing operations on control data, are connected using virtual wires. The layout and operation of a Pd program attempts to mimic a design and working principle of modular synthesizers. The controller program, presented in Figure 5.13, makes use of control data processing objects only. Such objects usually send an output value in reaction to an input value, or a set of values. Outputs are located on the bottom, and inputs – on the objects' top edge.

The program performs two kinds of operations: it handles network connections, and reads data produced by a controller. In presented case the controller is supposed to be a MIDI device equipped with a velocity-sensitive keyboard and a number of additional input devices such as rotary encoder knobs or sliders, that can be assigned to parameters of the instrument model. A value originating from a physical controller is wrapped into a control sequence consisting of a space-separated parameter symbol

and value. Such sequence is sent through a network to the host program running the synthesizer engine.

Bottom-right part of the patch contains connection and disconnection facilities. It is possible to set IP address of the server, as well as port address it listens to on. A 'localhost' is selected as a default. Once a 'termination' sequence is sent and received by the host program, it breaks the main loop, and allows to close the synthesizer.

The remaining groups of objects handle the MIDI device. Specific controller values assigned to `ctlin` (i.e. 'control in') objects are configured to comply with a particular physical device, and are to be adjusted in case of its change. The program reacts to key presses broken into two parameters, key index and velocity, as well as to pitch-bend status, and positions of nine additional controllers. Each parameter has a control sequence identifier assigned, which allows the host program to identify it.

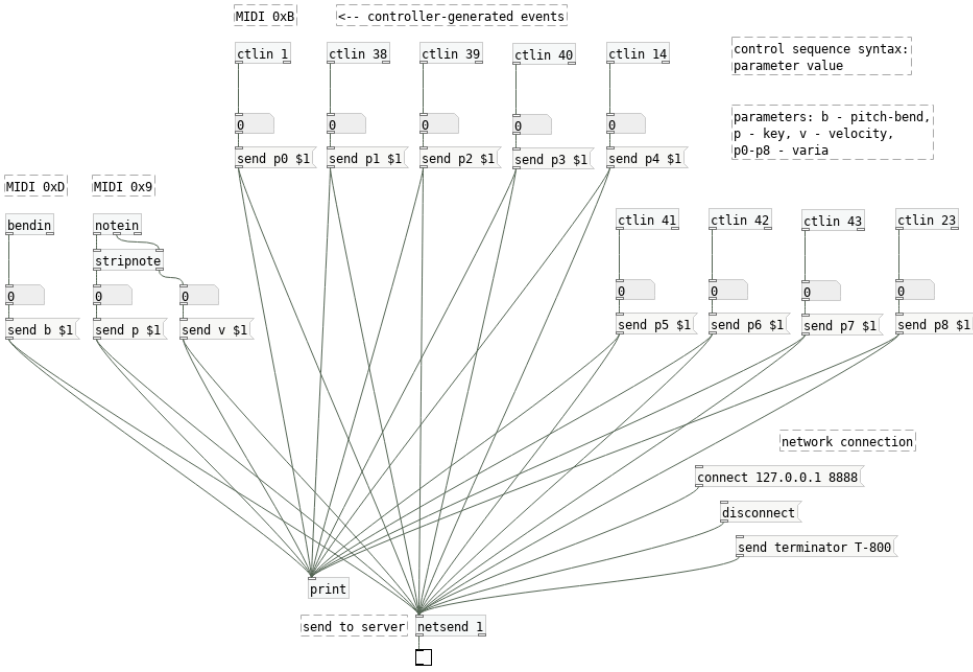


Figure 5.13. An implementation of the controller program as a PureData patch running under Pd-L2ork/Purr-Data distribution

5.3.4.3. Handling Control Events

The host program utilises a mechanism of network sockets and the UDP transport layer protocol. Unlike in case of TCP, an UDP server handles incoming data from all remote clients sequentially, through a single socket, without a need to create child processes. Therefore it is possible to distribute control over several controller programs, e.g. to handle spatially distributed control devices.

UDP server is started in the initialisation part of the host program, as shown in Listing 5.31. After required variables have been declared and initial values set, an UDP socket is created. Function `die` simply prints an error message and exits the program. Next, created socked is attempted to be bound to a port defined using `#define` preprocessor directive.

Listing 5.31. Starting UDP server in host program

```
// variables
int s;
char buf0[BUFLEN], buf1[BUFLEN], buf2[BUFLEN];
struct timeval tv;
fd_set readfds;
struct sockaddr_in si_me, si_other;
// set
tv.tv_sec = 0;
tv.tv_usec = 0;
// create UDP socket
if ((s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1) die("socket");
fcntl(s, F_SETFL, O_NONBLOCK);
// fill structure with zeros
memset((char *) &si_me, 0, sizeof(si_me));
// set
si_me.sin_family = AF_INET;
si_me.sin_port = htons(PORT);
si_me.sin_addr.s_addr = htonl(INADDR_ANY);
// bind socket to port
if( bind(s , (struct sockaddr*)&si_me, sizeof(si_me) ) == -1) die("bind");
// initialise socket descriptor
FD_ZERO(&readfds);
FD_SET(s, &readfds);
```

In each pass of the main loop host program checks for an incoming data using `select` function and handles it depending on its content, as shown in Listing 5.32. Data received to a text buffer is attempted to be parsed, i.e. divided into parameter identifier and value. If a special control string 'terminator' is found, the main loop is broken. Otherwise a series of checks follow, one of which is shown in the listing. Parameter `p3` is recognised as a readout position. Related kernel buffer is read, and readout position is changed for all strings – a seven-bit MIDI value is converted to a normalised floating point variable. Afterwards, modified array of parameters overwrites appropriate OpenCL buffer. Other parameters are adjusted in the same way.

Listing 5.32. Receiving and handling data

```
if (select(s+1, &readfds, NULL, NULL, &tv) == 1)
{
    if ((recv_len = recvfrom(s, buf0, BUFLEN, 0, (struct sockaddr *) ↩
        &si_other, &slen)) == -1) printf("Error receiving packet");
    if (sscanf(buf0, "%s %s" , buf1, buf2) == 2)
```

```
    {
      if (strcmp(buf1, "terminator") == 0) break;
      j = atoi(buf2);
      // check for readout position change
      if (strcmp(buf1, "p3") == 0)
      {
         if (clEnqueueReadBuffer(commandQueue, bufferI, CL_TRUE, 0, ↩
             (4+(nstr*2))*sizeof(cl_int), ipar, 0, NULL, NULL) != ↩
             CL_SUCCESS) printf("Error retrieving OpenCL buffer");
         for (i=0; i<nstr; i++)
         {
            rp[i] = ((cl_float)j)/127.0;
            rpi[i] = 1 + floor(GridSize[i]*rp[i]);
            ipar[5+(i*2)] = rpi[i];
         }
         if (clEnqueueWriteBuffer(commandQueue, bufferI, CL_TRUE, 0, ↩
             (4+(nstr*2))*sizeof(cl_int), ipar, 0, NULL, NULL)!=CL_SUCCESS) ↩
             printf("Error writing OpenCL buffer");
      }
      // remaining checks ...
    }
}
```

Readout position is one of parameters that require only a simple adjustment. Some parameters though affect fundamental properties of the model, which triggers a series of adjustments. One of such parameters is the string fundamental frequency. It directly affects γ value, which in turn is used to calculate stability condition. Sufficiently large change of the stability condition can alter value of spatial grid spacing X and grid size N. Consequences are considerable. New excitation distribution array has to be calculated. Current string grid has to be interpolated to the new one, with different number of grid points. In addition, a large set of parameters and simulation variables has to be recalculated, including loss parameters, readout location, scheme coefficients, and circular buffer offsets. Such procedure though allows a string to continuously sound during parameter adjustments, which is a desirable feature, allowing a user to perform various expressive parameter 'glides'.

5.4. Hyper-Dimensional Objects

While describing vibrating objects it is a common approach to start with a simple, one-dimensional case, and increase complexity by expanding it to two, and finally three dimensions. It allows to draw analogies through pointing out similarities and differences. In such manner a string may be compared to a membrane, etc. Obviously comparisons end with three spatial dimensions.

Yet a musician experimenting with physical modelling synthesis might ask, while designing models of strings or membranes: is it possible to extrapolate further? How

would it sound if a hypothetical membrane had three, or better yet – four dimensions? What kind of new sound features could emerge? Such question leads to the first, and possibly the most spectacular group of infeasible instruments: hyper-dimensional objects. Instruments of this group could not exist either by simply requiring more than three spatial dimensions, or because increasing their initial dimensionality would make them infeasible, such as a three-dimensional membrane.

5.4.1. Hyper-Membrane

One can define a N-dimensional hyper-membrane as a hypothetical object which:

- has $N + 1$ spatial dimensions, one of which is small, while the remaining N are large in relation to the wavelength,
- is clamped along all of its boundaries,
- is stretched in N 'large' dimensions, which is a source of its elasticity,
- is excited and vibrates along $(N + 1)$-st 'small' dimension.

Such definition clearly makes even a three-dimensional membrane an infeasible instrument, vibrating in a missing fourth spatial dimension. Moreover, due to required clamping on all boundaries it would be inaccessible for any external excitation, nor would it be able to radiate any vibration outside.

5.4.1.1. Basic Model

A mathematical N-dimensional hyper-membrane could be described simply by a N-dimensional wave equation. Using scaled variables it can be written as

$$\frac{\partial^2 u}{\partial t^2} = \gamma^2 \Delta_{\mathrm{ND}} u \tag{5.8}$$

or with a simple loss model

$$\frac{\partial^2 u}{\partial t^2} = \gamma^2 \Delta_{\mathrm{ND}} u - 2\sigma_0 \frac{\partial u}{\partial t} \tag{5.9}$$

The equation is identical to the two-dimensional case (3.262), apart from the Laplacian, which in a generalised N-dimensional case assumes the following form

$$\Delta_{\mathrm{ND}} u = \sum_{i=1}^{N} \frac{\partial^2 u}{\partial x_i^2} \tag{5.10}$$

where x_i are subsequent spatial coordinates, i.e. in two dimensional case $x_2 = y$, and the Laplacian assumes the form given in (3.69).

The model may be further expanded with features described in case of a string and a two-dimensional membrane. It is possible to simulate frequency-dependent loss, stiffness, or preparation through addition of springs and dampers. While theoretically possible, an expansion of a non-linear model into more than two dimensions would require considerably more effort, while audibility of its effects might be reduced by phenomena originating from multi-dimensional approximation. Therefore so far only the linear model has been implemented.

5.4.1.2. Excitation

Out of various ways to excite a hyper-membrane, striking is the most straightforward and intuitive. Plucking and bowing might be implemented as well, however their interpretation might be more confusing for a user of a synthesizer. As was the case in one and two dimensions, spatial distribution of an excitation, either u_0 or v_0, may be modelled with the raised cosine, which in N dimensions is given by

$$c_{rc}(\boldsymbol{x}) = \begin{cases} \frac{c_0}{2}\left(1 + \cos\left(\frac{\pi r_{\mathrm{ND}}}{r_{hw}}\right)\right) & r_{\mathrm{ND}} \leq r_{hw} \\ 0 & r_{\mathrm{ND}} > r_{hw} \end{cases} \tag{5.11}$$

where c_0 is the peak displacement, r_{hw} is the half-width of a pulse, and coordinates are given by the vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_N)$. The N-dimensional distance is expressed as

$$r_{\mathrm{ND}} = \sqrt{\sum_{i=1}^{N}(x_i - x_{0,i})^2} \tag{5.12}$$

where $\boldsymbol{x}_0 = (x_{0,1}, x_{0,2}, \ldots, x_{0,N})$ is the peak position.

5.4.1.3. Finite Difference Scheme

Moving from one or two dimensions to a generalised N-dimensional space, a spatial index of a grid function will have to be supplemented with a N-component vector. Therefore $u = u_{\boldsymbol{l}}[n] = u_{(l_1, l_n, \ldots, l_N)}$, and l_i is a spatial index along dimension x_i.

A model given by (5.9) can be approximated using the following explicit differential scheme

$$\delta_{tt} u_{\boldsymbol{l}}\ [n] = \gamma^2 \delta_{\Delta \mathrm{ND}} u_{\boldsymbol{l}}\ [n] - 2\sigma_0 \delta_{t\cdot} u_{\boldsymbol{l}}\ [n] \tag{5.13}$$

Approximation of the N-dimensional Laplacian $\delta_{\Delta \mathrm{ND}}$ may be based on the five-point operator used in two-dimensional cases (3.72), and use pairs of grid points adjacent to the centre along each axis. Assuming equal grid spacing X in all dimensions, it is expressed as

$$\begin{aligned} \delta_{\Delta \mathrm{ND}} u_{\boldsymbol{l}}\ [n] =& \sum_{i=1}^{N} \delta_{x_i x_i} u_{\boldsymbol{l}}\ [n] = \\ =& \frac{1}{X^2}(u_{(l_1+1, l_2, \ldots, l_N)}[n] + u_{(l_1-1, l_2, \ldots, l_N)}[n] + \\ &+ u_{(l_1, l_2+1, \ldots, l_N)}[n] + u_{(l_1, l_2-1, \ldots, l_N)}[n] + \ldots + \\ &+ u_{(l_1, l_2, \ldots, l_N+1)}[n] + u_{(l_1, l_2, \ldots, l_N-1)}[n] - 2N u_{(l_1, l_2, \ldots, l_N)}[n]) \end{aligned} \tag{5.14}$$

The scheme expands into

$$\begin{aligned}
&\frac{1}{T^2}(u_{(l_1,l_2,\ldots,l_N)}[n+1] - 2u_{(l_1,l_2,\ldots,l_N)}[n] + u_{(l_1,l_2,\ldots,l_N)}[n-1]) = \\
&= \frac{\gamma^2}{X^2}(u_{(l_1+1,l_2,\ldots,l_N)}[n] + u_{(l_1-1,l_2,\ldots,l_N)}[n] + \ldots + \\
&+ u_{(l_1,l_2,\ldots,l_N+1)}[n] + u_{(l_1,l_2,\ldots,l_N-1)}[n] - 2Nu_{(l_1,l_2,\ldots,l_N)}[n]) - \\
&- \frac{\sigma_0}{T}(u_{(l_1,l_2,\ldots,l_N)}[n+1] - u_{(l_1,l_2,\ldots,l_N)}[n-1])
\end{aligned} \tag{5.15}$$

which results in the following recursion

$$\begin{aligned}
u_{(l_1,l_2,\ldots,l_N)}[n+1] =& \frac{2\left(1-N\lambda^2\right)}{\sigma_0 T+1}u_{(l_1,l_2,\ldots,l_N)}[n] + \\
&+ \frac{\lambda^2}{\sigma_0 T+1}(u_{(l_1+1,l_2,\ldots,l_N)}[n] + u_{(l_1-1,l_2,\ldots,l_N)}[n] + \\
&+ \ldots + u_{(l_1,l_2,\ldots,l_N+1)}[n] + u_{(l_1,l_2,\ldots,l_N-1)}[n]) + \\
&+ \frac{\sigma_0 T-1}{\sigma_0 T+1}u_{(l_1,l_2,\ldots,l_N)}[n-1]
\end{aligned} \tag{5.16}$$

A future grid point value is calculated using $(2N+2)$ points from two time steps: $(2N+1)$ from the current step, i.e. central and nearest neighbours along each axis, and a central one from the previous step. Calculations require three coefficients: two separate for both central points, and one common for all neighbours.

5.4.1.4. Stability

In case of a model without loss (5.8), a stability condition obtained through von Neumann analysis [61] would be given by

$$\lambda \leq \frac{1}{\sqrt{N}} \quad \rightarrow \quad X \geq \gamma T\sqrt{N} \tag{5.17}$$

which allows to easily determine spatial grid spacing depending on the dimensionality N.

If a loss is introduced to the model through addition of parameter σ_0 (5.9), use of the following ansazt

$$u_{\boldsymbol{l}}\,[n] = z^n e^{iX\sum_{j=1}^{N} l_j \beta_{x_j}} \tag{5.18}$$

allows to write the characteristic polynomial for the scheme (5.16)

$$z + \frac{1}{\sigma_0 T+1}\left(4\lambda^2\left(\sum_{i=1}^{N} p_{x_i}\right) - 2\right) + \frac{1-\sigma_0 T}{\sigma_0 T+1}z^{-1} = 0 \tag{5.19}$$

where $p_{x_i} = \sin^2(\frac{1}{2}\beta_{x_i}X)$. Roots of the polynomial are bounded by unity under the same condition as in the case without loss (5.17).

5.4.2. Model Implementation

5.4.2.1. Implementation Considerations

A general design and operation of a hyper-membrane synthesizer is similar to the OpenCL implementation of the string model. A host program coordinates memory management, dispatches kernels, handles playback, and receives control from an external module. A kernel runs the actual simulation. However, there are some significant differences.

The first concern is the size of the N-dimensional grid. Its initialisation is presented in Listing 5.33, where the size is derived from the grid shape defined by a user through setting domain aspect ratios `epsilon` related to the first dimension x_1. As was the case of string, grid spacing and size can be calculated using the stability condition. Variable `ND` in the listing is the number of dimensions. `T` and `X` are the temporal and spatial grid spacing, respectively.

Listing 5.33. Setting grid size and stability-related parameters

```
cl_float lambda = 1.0/sqrt((cl_float)ND); // Courant number
cl_float X = gamma*T/lambda; // initial grid spacing
int *Nx; // number of xn-subdivisions of spatial domain
Nx = (int *)malloc(ND*sizeof(int));
Nx[0] = (int)floor(sqrt(epsilon[0])/X);
for (i=1; i<ND; i++) Nx[i] = (int)floor(1.0/(sqrt(epsilon[i-1])*X));
X = sqrt(epsilon[0])/((cl_float)Nx[0]);
lambda = gamma*T/X; // recalculate Courant number
int *SX; // domain size in each of coordinates
SX = (int *)malloc(ND*sizeof(int));
for (i=0; i<ND; i++) SX[i] = Nx[i]+1;
int DS = SX[0]; // total number of elements in domain
for (i=1; i<ND; i++) DS *= SX[i];
int *XN; // temporary storage for N-dimensional coordinates
XN = (int *)malloc(ND*sizeof(int));
```

The OpenCL allows to implement parallel algorithms in multidimensional domains as long as the number of dimensions does not exceed three. It is limited by the arrangement of the NDRange. Therefore in order to implement an instrument model with adjustable and not arbitrarily limited number of dimensions it was necessary to design storage facility able to handle multiple dimensions, but fitting low-dimensional memory buffers.

One-dimensional array is the most convenient, considering passing it as an argument to a function, and the most straightforward when converting from higher-dimensional arrays. Thus it has been chosen to store grid data. Two functions, presented in Listings 5.34 and 5.35, allow to convert between coordinates in N-dimensional space and its flat, 1-dimensional representation. Arguments include the number of dimensions, and a grid size stored in an array. Multi-dimensional coordinates are stored in an array as well.

Listing 5.34. Function performing conversion from N-dimensional to 1-dimensional coordinates

```
static inline int to1D(int x[], int xdim[], int N)
{
   // x[]     coordinates (x1,x2,...,xN)
   // xdim[]  size of grid (x1len,x2len,...,xNlen)
   // N    number of dimensions
   int i, xlin = x[0], dim = 1;
   for (i=1; i<N; i++)
   {
      dim *= xdim[i-1];
      xlin += x[i]*dim;
   }
   return xlin;
}
```

Listing 5.35. Function performing conversion from 1-dimensional to N-dimensional coordinates

```
static inline void toND(int xlin, int x[], int xdim[], int N)
{
   // xlin  position in 1D array
   // x[]     returned coordinates (x1,x2,...,xN)
   // xdim[]  size of grid (x1len,x2len,...,xNlen)
   // N    number of dimensions
   int i, xtmp = xlin, dim = 1;
   for (i=0; i<N; i++) dim *= xdim[i];
   for (i=N-1; i>=0; i--)
   {
      dim /= xdim[i];
      x[i] = xtmp/dim;
      xtmp %= dim;
   }
   return;
}
```

In order not to invoke the same calculations multiple times during kernel iterations, an array containing 1-dimensional coordinates of neighbouring points along all dimensions is prepared beforehand, as presented in Listing 5.36, at the expense of additional memory usage.

Listing 5.36. Preparation of the array of neighbours

```
cl_int* coord = (cl_int*)malloc(2*ND*DS*sizeof(cl_int));
for (i=0; i<DS; i++)
{
   toND(i,XN,SX,ND);
   for (j=0; j<ND; j++)
```

```
   {
      XN[j] = XN[j]-1;
      coord[(2*ND*i)+(2*j)] = to1D(XN,SX,ND);
      XN[j] = XN[j]+2;
      coord[(2*ND*i)+(2*j)+1] = to1D(XN,SX,ND);
      XN[j] = XN[j]-1;
   }
}
```

The primary use of the `coord` array is to obtain one-dimensional addresses of points belonging to the stencil, while updating the grid point value according to FD scheme (5.37). Variables `c0`–`c2` store grid offsets of next, current, and previous time steps.

Listing 5.37. Use of the array of neighbours in the kernel

```
tmp = 0.0;
for (j=0; j<ND; j++)
{
   k = (2*ND*i)+(2*j);
   tmp += u[c1+coord[k]] + u[c1+coord[k+1]];
}
u[c0+i] = (s1*tmp) + (s0*u[c1+i]) + (t0*u[c2+i]);
```

The final concern regards the relation of total number of grid points and work-group size. In all practical cases the former will be larger than the latter. Therefore it is not possible to directly apply approach from the string case, where the kernel was internally synchronising work-items and calculating a whole audio buffer in a single run. There are two solution, and either one can be applied depending on the architecture and processing capabilities of the OpenCL device used for simulation.

According to the first solution each work-item calculates not one, but a larger number of grid point values, consisting a subset of a spatial grid. Number of points to calculate by a work-item is determined so as to limit number of work-items to a single work-group, thus allowing for internal synchronisation. The second solution involves external synchronisation. In this case number of work items matches the number of grid points, preventing internal synchronisation, but a single run of the kernel calculates only a single time step, after which an external synchronisation is performed by returning to the host program.

The number of grid points grows exponentially while adding subsequent dimensions, preventing from calculating higher-dimensional grids in real time. However, it is realistic to expect higher capabilities for parallel processing in near future. Due to problems with increasing clock frequencies, new generations of processing devices are equipped with larger numbers of parallel units instead. New, massively parallel GPUs are goaled much more towards general purpose computing, than graphics processing only.

Before a sufficient hardware is available, it is possible to further increase dimensionality in two ways. The first one is to sacrifice sound quality by either decreasing sampling frequency or by increasing grid spacing only. Both lead to degradation in

higher frequency region of produced signal spectra, yet they significantly reduce number of grid points. The second way is to investigate possibilities of synchronisation across work-groups in OpenCL. By design, such synchronisation is not guaranteed, mostly due to a diversity of underlying hardware layer. There are attempts though, such as work of Sorensen et al. [529], to provide such synchronisation at a cost of more restricted compatibility, i.e. to GPUs only. The method ports the XF barrier [612] to the OpenCL, and utilises a master/slave model, where threads of a single master work-group manage slave work-groups.

5.4.2.2. User-Controllable Instrument Parameters

A hyper-membrane implemented according to model (5.9) is a relatively abstract instrument. Being based on a wave equation with a simple addition of a loss parameter, it has a limited set of properties that can be adjusted. However, it has a unique advantage of ability to define and control its geometry.

A complete list of parameters a user can control is presented in Table 5.3. Some features are described not by a single value, but by a vector. Such is the case of readout location and location of excitation, both of which have to be sets of coordinates in a N-dimensional space. An interesting case is the aspect ratio, which is the primary timbre control mechanism. The shape of a domain is a hyperrectangle, also referred to as a n-orthotope [141]. The aspect ratio relates grid size along each axis to the first spatial dimension x_1, thus it has $(N-1)$ components defining dimensions x_2–x_N.

Table 5.3. Synthesis-related simulation parameters that may be controlled by a user in the hyper-membrane simulator

Parameter	Type	Components	Real-time
Sample rate [Hz]	`int`	1	No
Number of spatial dimensions N	`int`	1	No
Domain aspect ratios related to x_1	`float`	$N-1$	Yes
Wave speed γ	`float`	1	Yes
Loss parameter T_{60} [s]	`float`	1	Yes
Location of excitation centre (normalised)	`float`	N	Yes
Width of excitation (normalised)	`float`	1	Yes
Initial displacement	`float`	1	Yes
Initial velocity	`float`	1	Yes
Readout location (normalised)	`float`	N	Yes

5.4.3. Example Signals

Figures 5.14 and 5.15 present examples of signals produced by hyper-membranes using a parallel OpenCL implementation described in a previous subsection. The main purpose is to observe the effect of geometry on the signal features. Due to effects manifesting in both, spectral and temporal structure of a signal, data is presented in

a form of spectrograms. For the sake of comparison, other than geometric parameters of simulations have been fixed, and are common for all examples. They are given in Table 5.4.

Table 5.4. Common values of simulation parameters for signals presented in Figures 5.14 and 5.15

Parameter	Value	Comment
Sample rate [Hz]	32 kHz	—
Wave speed γ	440	—
Loss parameter: T_{60} [s]	8	—
Normalised location of excitation centre	0.21	Same for all dimensions
Normalised width of excitation	0.15	—
Initial displacement	0	—
Initial velocity	1	—
Normalised readout location	0.3	Same for all dimensions

Figure 5.14 illustrates changes introduced by increasing a number of dimensions in two cases. Membranes presented in the left column have a hyper-cubical shape, i.e. all their aspect ratios are set to 1. Membranes in the right column have different, randomly chosen ratios for each pair of dimensions, i.e. $x_2 : x_1 = 1.91$, $x_3 : x_1 = 2.21$, $x_4 : x_1 = 1.47$, and $x_5 : x_1 = 4.63$. In both cases two effects dominate – one is directly caused by geometry, while the other is the result of implementation.

The first effect is purely spectral. With increasing number of dimensions the pattern of reflections grows in complexity, and it is more difficult to isolate individual partials. In low-dimensional membranes they are less numerous and more distinct. In four or five dimensions the energy is spread more evenly across spectrum. There are more weaker, densely distributed partials. Perceptually, low-dimensional membranes tend to invoke a stronger sensation of pitch or pitches not unlike a multiphonics effect. Higher-dimensional membranes produce sounds with higher content of a noise-like component. Frequencies and amplitudes of partials are affected in a general manner by domain aspect ratios, which can be observed by comparing spectrograms in left and right column of Figure 5.14. The ratio is a convenient parameter for a user, who can adjust spectral structure changing geometry in real-time, and using auditory feedback. One might attempt to obtain detailed signal features by finding analytic solutions for the model, though the problem becomes significantly more complex in higher dimensions. Moreover, such solutions would be inaccurate due to properties of model implementation.

The implementation is the source of the second effect, which grows particularly salient in higher-dimensional membranes, and causes the result of simulation to divert from a purely analytical solution, or from a real object behaviour, if such object could exist at all. The effect manifests itself in spectrograms as a series of curved arcs. Each arc represents an individual reflection that has been warped by a strong numerical dispersion.

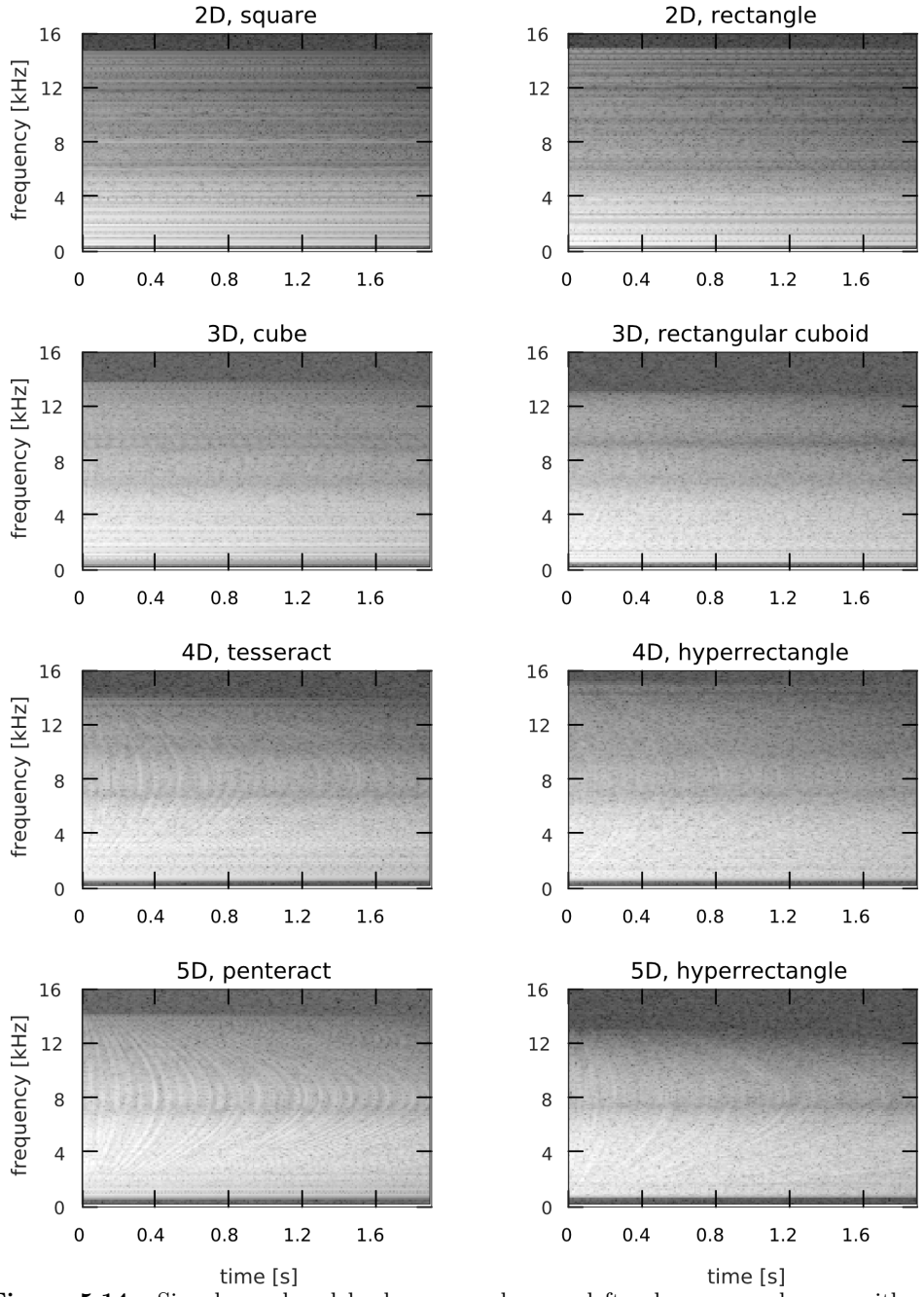


Figure 5.14. Signals produced by hyper-membranes; left column: membranes with all aspect ratios equal; right column: different aspect ratios – details are given in text

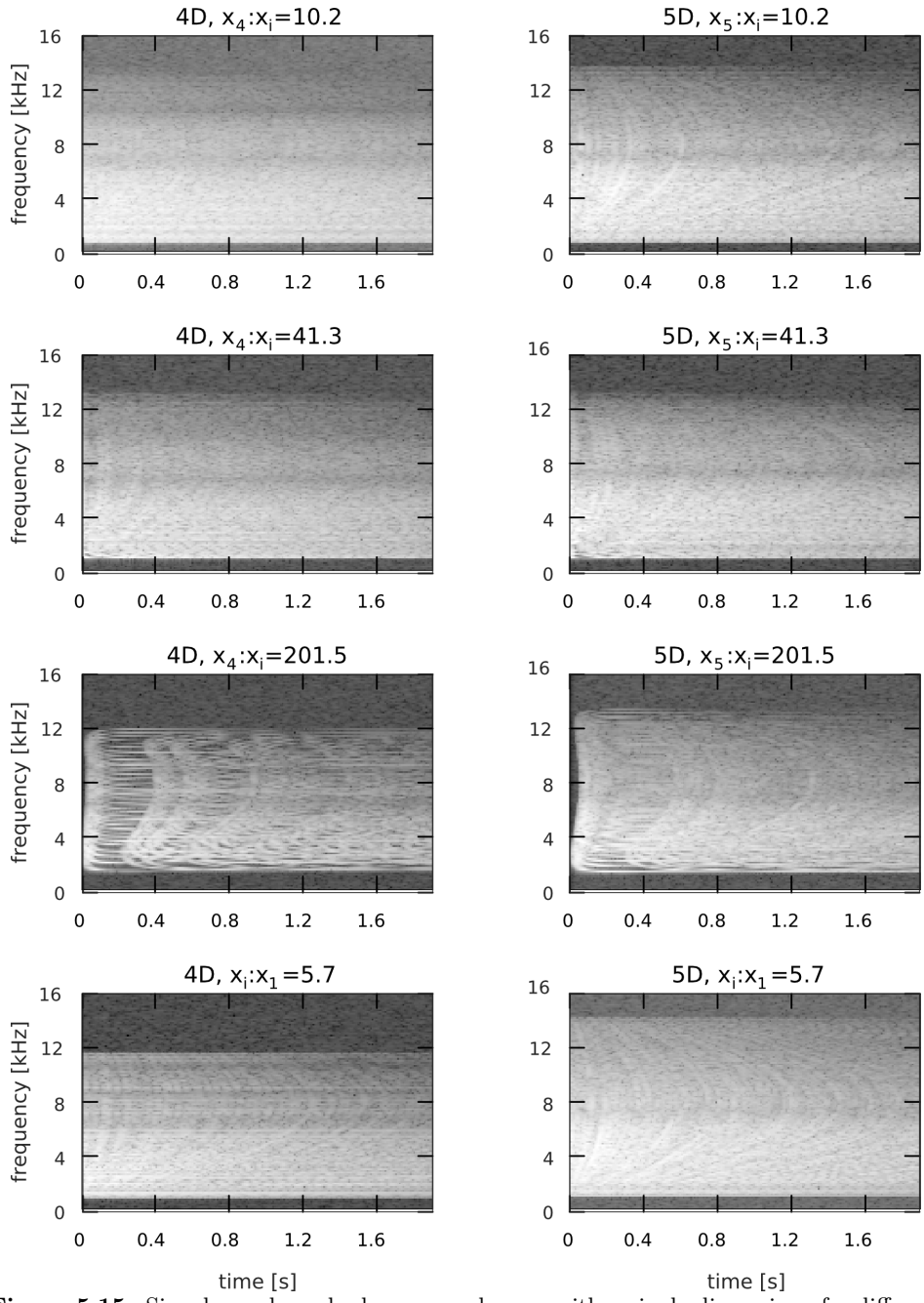


Figure 5.15. Signals produces by hyper-membranes with a single dimension of a different size than the rest – details are provided in text

Dispersive effects may appear even in two-dimensional cases, and are traditionally regarded as undesirable phenomena due to distortions caused in produced signal. Thus they are attenuated by applying different approximations for the Laplacian operator [61], e.g. a parametrised form (3.74) with a parameter adjusted in some optimisation procedure, instead of a form using adjacent grid points only (3.72). This however, leads to significant increase in computations required to update a single grid point in higher-dimensional models. While the simpler Laplacian approximation uses only $(2N + 1)$ grid points, the parametrised form needs as much as 3^N. The former grows linearly, the latter – exponentially. In two dimensions the difference is 5 and 9 points, but in five dimensions it jumps to 11 and 243. In infeasible instruments however, the consequence of dispersion shall not be regarded as undesirable. The reason is not only a computational difficulty of its attenuation, but primarily a remarkably intriguing auditory effect manifesting in higher-dimensional membranes. It may be regarded as a case of emergence [533], where simple effects normally leading to distortions, in a higher dimensional space produce complex structures in a spectro-temporal plane.

The emergence is particularly pronounced in a case illustrated by Figure 5.15, where each membrane, either four or five-dimensional, has a single dimension that largely differs in size from the remaining ones. First three rows represent membranes that have all dimensions equal, except for the last, larger one. Its aspect ratio relative to the others is indicated over each particular chart. The remaining ratios are set to 1. In the last row the situation is different: all aspect ratios are equal and set to 5.7, therefore a single dimension is smaller than all others.

A particularly interesting phenomenon emerges when a single dimension is larger than others. When a difference is small, dispersive arcs begin to show, stronger in five dimensions. They become more distinct with increasing ratio, and finally in case of the largest presented difference, arcs turn into a very complex, braid-like structure audible as a rhythmic series of decaying pulses morphing into multi-pitch glides. The structure is controllable through adjustment of aspect ratios. The last row case, with a single dimension smaller, results in a mixture of arc-evoked glide effects and a fast series of reflections, on the threshold of asperity. Again, it is more pronounced in a five-dimensional variant.

5.4.3.1. Brief Evaluation

With several dimensions to manipulate, it is possible to combine reflection and dispersion related effects, while adjusting frequencies and shapes of both. Yet, a detailed relation between controllable values and effect characteristics remains to be established. It is however already apparent, that hyper-membranes combine two sets of features. Firstly, due to physical modelling synthesis background, they behave predictably, e.g. with regards to excitation parameters. They can be struck either in the middle, or near the edge, which would result in an auditory effect expected by a musician. Change in excitation width brings intuitive spectral changes as well. Secondly, due to emergent behaviour of multi-dimensional dispersion, hyper-membranes tend to produce complex spectro-temporal structures, which are controllable through adjustment of geometric properties of a model. These two sets of features make the hyper-membranes interesting candidates for performance instruments.

5.4.4. Other Instruments

A membrane simulated using a wave equation is one of the simplest models that may be regarded as a musical instrument, once it has been supplemented with a proper control mechanism. Its simplicity, in case of Cartesian coordinates and Laplacian approximation given by (5.10), allows for relatively easy introduction of additional spatial dimensions which leads to emergence of new features. There is a cost of such simplicity though, and it is a limited set of physically-relevant instrument or performance parameters. Therefore, once enough computing power becomes available, it shall be interesting to test more sophisticated models.

In presented case a very basic membrane excitation mechanism has been applied, without actual simulation of the excitation element, and coupling it to the membrane. Such mechanism is convenient for observation of model properties, allowing to analyse features of a hyper-membrane alone. On the performance side, it provides a basic set of parameters, such as excitation width, location, initial displacement, and initial velocity, thus it can simulate both pluck and strike. However, different excitation mechanisms shall provide much wider set of parameters, and could introduce finer details into the signal. It is possible to apply a model of coupled hammer, as described in case of the string simulation program. As a lumped element, it does not depend on the membrane dimensionality. Excitation through bowing may be simulated in a similar way. Higher number of dimensions provides more opportunities to experiment with spatial distribution of excitation, such as various multi-point configurations. Moreover, one might vary excitation area, shape, and dimensionality.

In case of a string it was straightforward to improve a model by adding frequency-dependent loss and stiffness. The former is possible, although due to spatial dependence of the effect it requires use of the Laplacian combined with time derivative operator, expanding stencil of a scheme. Introduction of stiffness may be achieved by a simple expansion of Kirchhoff model (3.280) to arbitrary number of dimensions. However, it involves use of the biharmonic operator, which would further expand a scheme stencil. A stencil increases either linearly or exponentially with dimensionality, depending on assumed approximations. Therefore, even though both improvements are possible, it has to be considered that they effectively reduce achievable dimensionality and grid size that can be simulated due to increasing amount of calculations required for updating each grid point.

While multi-dimensionality alone allows hyper-membranes to produce signals with interesting features, in all presented variants the instrument modelled was a hyper-rectangle. Other shapes, either simple and regular, or including finer details, shall lead to new signal features. The first choice would be to switch to a different coordinate system, e.g. a spherical one. In N-dimensional space its coordinates include radius and $(N-1)$ angular coordinates, $(N-2)$ of which range over $[0, \pi]$, and one ranges over $[0, 2\pi]$. Such system would be convenient for modelling of hyperspheres, also referred to as n-spheres. Another choice might be a n-dimensional cylindrical coordinate system.

In a different approach one might attempt to set various types of boundary points not only on the edges of a domain, but in arbitrary positions. While this could

raise legitimate doubts regarding mathematical and physical formality of such procedure, particularly with jagged edges produced by angular or curved borderlines in sparse grids, it could provide a coarse way to delimit an arbitrary shape. Actually, such functionality had already been implemented in the hyper-membrane program through addition of a *boundary mask*. It is an array accompanying the grid, of the same size and shape. Its values determine whether a certain grid point can vibrate, or is clamped. Before updating a grid point, kernel simply checks the mask, and performs update only if the point is not clamped. In cases of attempts to reproduce finer shapes, such procedure shall produce considerably distortions, yet for simpler, more general shapes and larger grids, it may lead to useful results. Arbitrary shapes are considered to be handled with better results using different numerical methods, such as finite element method (FEM). It is difficult, however, to generalise FEM into higher-dimensional spaces, particularly regarding initial mesh generation. The problem requires further study. Once solved however, it will allow to freely shape and play any conceivable multidimensional instrument.

5.5. Impossible Boundaries

When a model of an instrument is designed, a suitable grid function and boundaries are chosen in attempt to reproduce behaviour of the physical object. Yet modelling facilities, consisting of data structures and implementations of algorithms for their processing, may be approached in a less constrained manner, allowing to design infeasible instruments. One example of such approach are hypothetical hyper-dimensional instruments discussed in the previous section. In a different approach one might change a behaviour of domain boundaries. In simulations of existing instruments variants of clamped, free, or supported boundary conditions are usually implemented. In a hypothetical model however, any kind of boundary behaviour might be considered, even though only some could produce interesting, useful musical effects. Among many possibilities, looping of a boundary can be a foundation of several promising properties, and will be discussed further.

5.5.1. Looped Boundaries

Three kinds of boundary behaviour have been studied, applied to membranes in Cartesian coordinates:

- a bi-directional loop,
- a one-directional loop,
- a loop combined with a twist.

All these types have a common property: a boundary grid point is not considered boundary. Instead, it has some other boundary grid point set as its neighbour in this particular direction.

5.5.1.1. Bi-Directional Loop

If a grid size in two dimensions p and q is given by N_p and N_q, a grid point value is given by u_{l_p,l_q}, and coordinates along these dimensions assume values $l_p \in [0, N_p - 1]$ and $l_q \in [0, N_q - 1]$, then a bi-directional loop along a dimension p is applied by the following substitution

$$\begin{aligned} u_{(-1,l_q)} &\triangleq u_{(N_p-1,l_q)} \\ u_{(N_p,l_q)} &\triangleq u_{(0,l_q)} \end{aligned} \tag{5.20}$$

If $N_p = N_q$, it is possible to set a perpendicular loop

$$\begin{aligned} u_{(-1,l_q)} &\triangleq u_{(l_q,N_q-1)} \\ u_{(l_p,N_q)} &\triangleq u_{(0,l_p)} \end{aligned} \tag{5.21}$$

If $N_p \neq N_q$ a perpendicular loop would require an interpolation.

Out of all looped boundaries, a bi-directional type is the closest to real physical objects. Tension and stiffness apart, ring-shaped idiophones, though not widely popular, might be considered as having such topology.

5.5.1.2. One-Dimensional Loop

Under the same assumptions, a one-directional loop in grid u_{l_p,l_q} would be defined by choosing only a single substitution from (5.20), and not applying the remaining one. Thus waves crossing one edge would enter the grid on the opposing side, yet waves reaching the opposite edge would encounter a normal boundary, and e.g. be reflected. Such behaviour is unlikely to be encountered in macroscopic physical objects made of conventional materials, like musical instruments.

5.5.1.3. Twisted Loop

While looping a grid along dimension p, a coordinate l_q may be twisted. It is accomplished using the following substitution

$$\begin{aligned} u_{(-1,l_q)} &\triangleq u_{(N_p-1,N_q-1-l_q)} \\ u_{(N_p,l_q)} &\triangleq u_{(0,N_q-1-l_q)} \end{aligned} \tag{5.22}$$

In case of two-dimensional grids it would produce the Möbius strip. It has an interesting property: being a one-sided surface, it has only a single boundary, and double the length of the original, not twisted strip.

5.5.2. Implementation Details

Certain design choices in the hyper-membrane implementation make it well suited to simulate looped boundaries with the wave equation almost directly, with only a minor effort to define a grid loop. The program does not access neighbouring grid points directly. Instead, it determines their coordinates for each grid point beforehand,

and stores them in array of neighbours `coord`, as shown in Listing 5.36. Therefore, one only needs to modify neighbour coordinates for selected boundary points. Such operation, for a bi-directional twisted loop, is performed by the code in Listing 5.38, which is inserted into the outer `i`-loop in the Listing 5.36. Grid coordinates in the program range from 1 to `SX[j]-2`.

Listing 5.38. Creating a twisted loop by modification of neighbour coordinates for boundary grid points

```
j = 0; // dimension
if (XN[j]==1)
{
   XN[j] = SX[j]-2;
   XN[j+1] = SX[j+1]-1-XN[j+1]; // twist
   coord[(2*ND*i)+(2*j)] = to1D(XN,SX,ND);
   XN[j+1] = SX[j+1]-1-XN[j+1];
   XN[j] = 1;
}
if (XN[j]==SX[j]-2)
{
   XN[j] = 1;
   XN[j+1] = SX[j+1]-1-XN[j+1]; // twist
   coord[(2*ND*i)+(2*j)+1] = to1D(XN,SX,ND);
   XN[j+1] = SX[j+1]-1-XN[j+1];
   XN[j] = SX[j]-2;
}
```

5.5.3. Selected Examples

A set of looped grids have been implemented as parallel OpenCL programs and used to produce acoustic signals. Examples can be divided into three groups, depending on the grid dimensionality and shape. In the first group, grid was a two-dimensional rectangle, with aspect ratio $x_2 : x_1 = 1.2$. In the second, it was a two-dimensional square. In the last one, grid was a three-dimensional rectangular cuboid, with aspect ratios $x_2 : x_1 = 1.2$ and $x_3 : x_1 = 1.5$. The remaining simulation parameters were set according to Table 5.4, with the exception of sampling rate, which was set to 44100 Hz.

5.5.3.1. Rectangle

Implemented rectangular grids are presented and described in Figure 5.16. A 'regualar' variant contains no loops, and has simple clamped boundaries. Spectra of signals produced by all implemented rectangular grids are shown in Figure 5.17. Legends in charts refer to particular grid variants as enumerated in Figure 5.16, for instance, 'Rd' stands for a rectangular grid with one-directional loop.

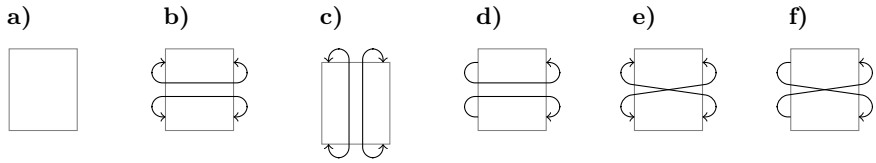


Figure 5.16. Rectangular grids: a) regular; b) bi-directional loop along x_1; c) bi-directional loop along x_2; d) one-directional loop along x_1; e) twisted bi-directional loop along x_1; f) twisted one-directional loop along x_1

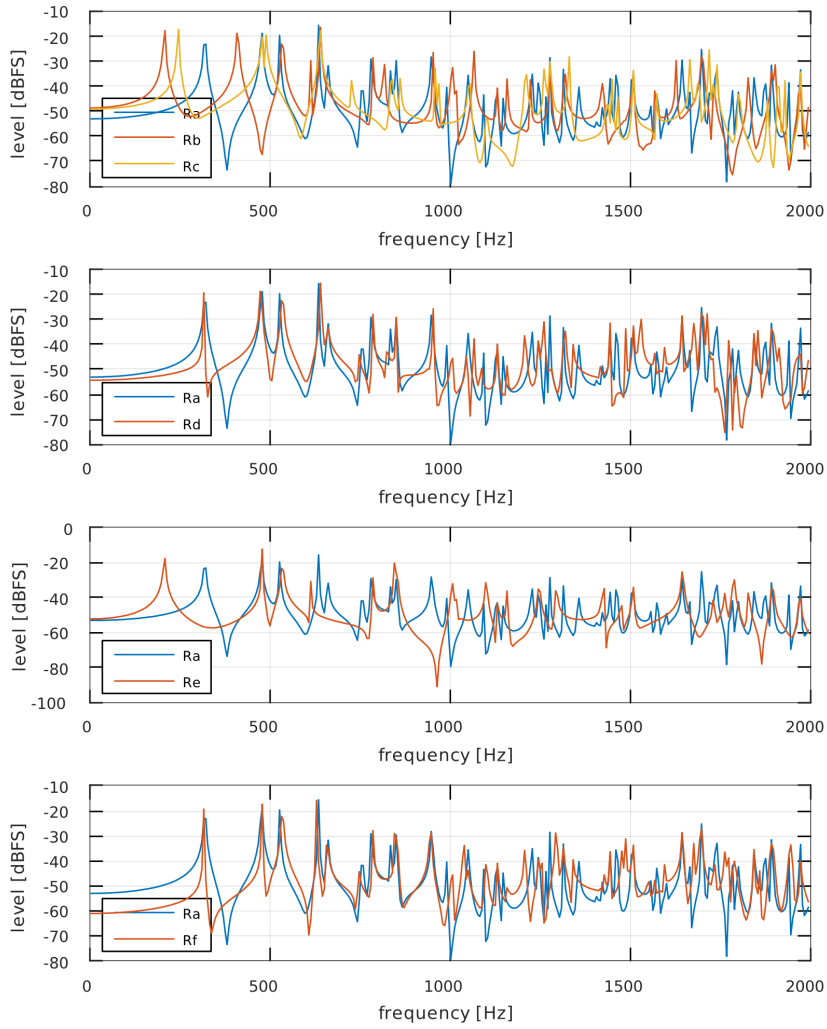


Figure 5.17. Spectra of signals produced by rectangular grids with loops; symbols in legends refer to rectangular (R) grid variants as enumerated in Figure 5.16

The most salient effect of applying a bi-directional grid loop is shift of part of a spectral structure towards lower frequencies, while partials related to reflections from unmodified boundary remain in place. This effectively corresponds to expanding the grid in one dimension, although at virtually no computational cost. A two-dimensional grid with two loops that can be enabled or disabled on demand is an effective way of achieving control of pitch in a membranophone or idiophone model with a single vibrating element. Compared to a loop alone, twisting a bi-directional loop leaves the lowest partial in place, and does not alter an overall spectral envelope, but changes the remaining partials. In effect, both loop variants are perceived as having the same pitch, but different timbre, albeit of a comparable brightness. A one-directional loop has much more subtle effect on a general spectral structure. Compared to a grid without loop, it does not alter the lowest partial, and slight spectral changes only start to appear in higher partials. The signal however acquires a very distinct component, i.e. a temporal structure of dispersion-bound curved arches, manifesting primarily on both sides of the quarter of sampling frequency. Therefore, while a bi-directional loop may be applied as a pitch-switch, and a twisted loop may serve as a timbre switch, then a one-directional loop can be applied as an effect adding a specific 'texture' on a spectro-temporal plane.

5.5.3.2. Square

A square grid has been implemented in order to study the effect of a perpendicular loop. Therefore only three square grid variants are compared, as shown in Figure 5.18. Spectra of signals produced by square grids are shown in Figure 5.19. Again, legends in charts refer to particular grid variants in Figure 5.18, for instance, 'Sa' stands for a regular square grid.

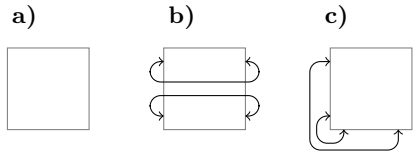


Figure 5.18. Square grids: a) regular; b) bi-directional straight loop; c) bi-directional perpendicular loop

The effects discussed in case of a rectangular grid are valid in case of a square one as well. It is worth noticing, that a straight bi-directional loop in a square membrane makes the lower part of a spectrum almost harmonic. Consequently, a pitch is not only getting lower, but also much more pronounced. A perpendicular bi-directional loop shifts parts of spectrum, including the lowest partial, towards lower frequencies. Moreover, a structure of several distinct, quasi-harmonic partials in lower parts of spectrum is no longer present, with only a few stronger partials remaining. As a consequence pitch saliency weakens as well.

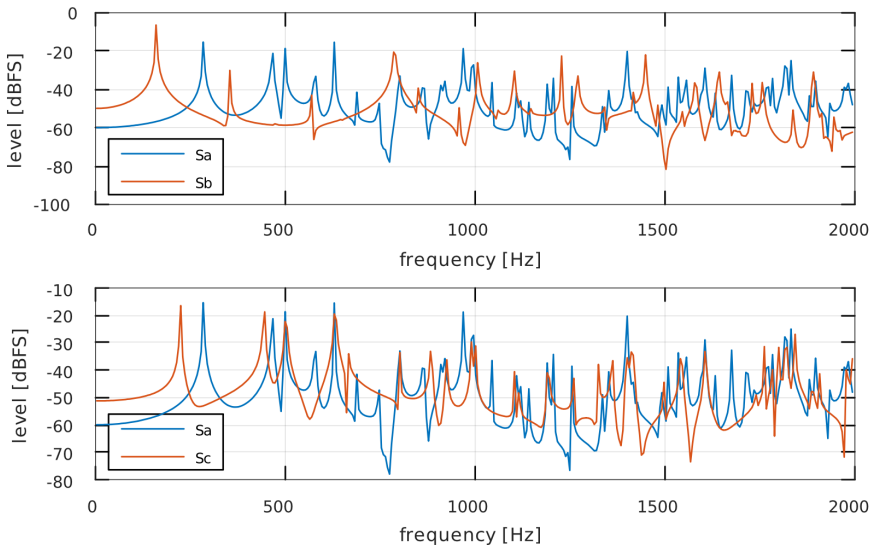


Figure 5.19. Spectra of signals produced by square grids with loops; symbols in legends refer to square (S) grid variants as enumerated in Figure 5.18

5.5.3.3. Rectangular Cuboid

A three-dimensional grid provides more combinations of places where various loops could be applied. The following cases have been selected for implementation:

a) regular,

b) bi-directional loop along x_1,

c) bi-directional loops along x_1 and x_2,

d) bi-directional loop along x_1 and one-directional loop along x_2,

e) one-directional loop along x_1 and bi-directional loop along x_2,

f) one-directional loops along x_1 and x_2,

g) twisted bi-directional loops along x_1 and x_2.

Spectra of signals produced by grids in question are shown in Figure 5.20. Legends in charts refer to particular grid variants from the list.

Principles observed in case of a two-dimensional rectangular grid are still valid in a three-dimensional rectangular cuboid. However, there are more opportunities to combine different features, such as lowering pitch through straight bi-directional loop along one dimension, and adding spectro-temporal arches texture by applying one-dimensional loop along another one. Two such textures with a different period may be combined using two one-dimensional loops.

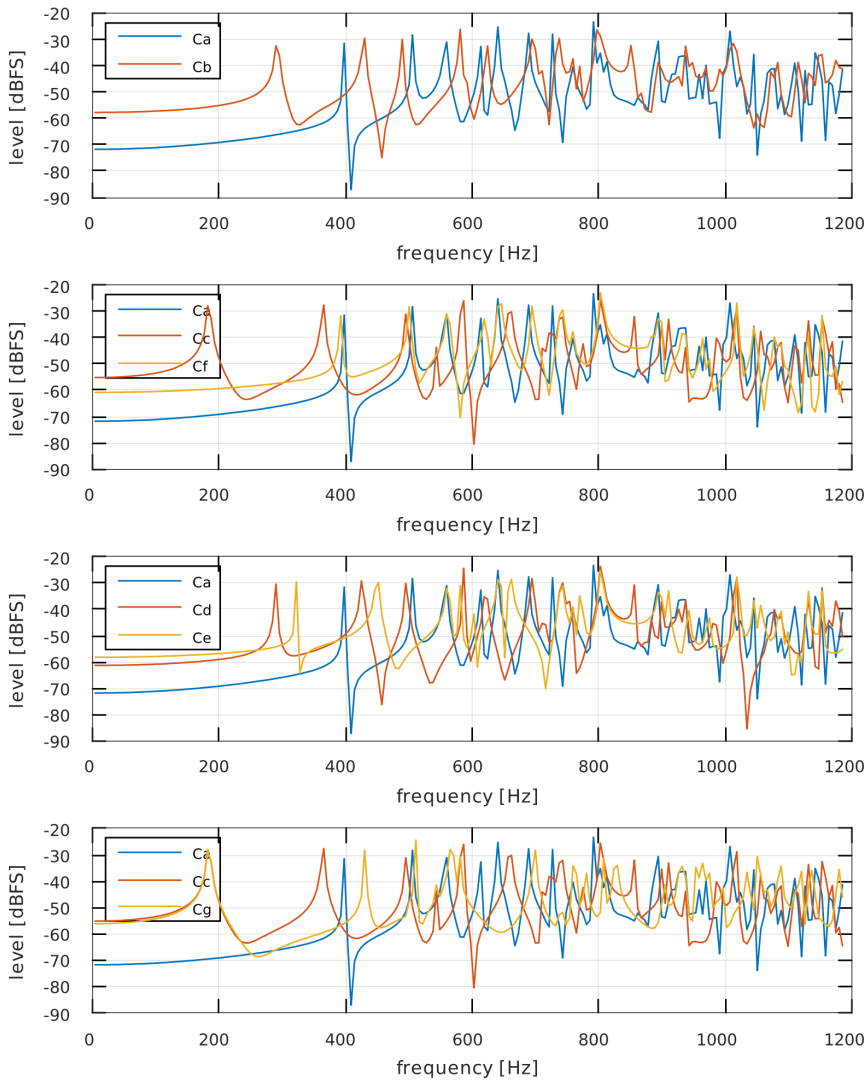


Figure 5.20. Spectra of signals produced by three-dimensional grids with loops; symbols in legends refer to cuboid (C) grid variants as enumerated in text

5.5.4. Further Study

Examples presented in the previous subsection represent only a small subset of cases where a grid boundary has been looped. Even these cases can be studied further in order to determine exact principles governing observed behaviour. Looped boundaries may be combined not only with clamped, but with other kinds of boundary

conditions as well. Some effects might emerge only in cases of particular dimensionality or aspect ratios.

As a different approach, instead of looping a grid, one might attempt to design and test entirely different boundary conditions, unlike these applied in models of physical objects. Interesting models could be designed using different coordinate systems as well. The most promising approach however, would be to carry out various topological experiments by combining loops, hyper-dimensionality, and arbitrary boundary locations.

Nevertheless, even a limited set of presented scenarios allows to supplement design of infeasible instruments with a number of tools which have known properties when applied to a wave equation approximated using a finite difference method. They can be immediately implemented in sound synthesizers based on FD method to enhance model-altering capabilities.

5.6. Evolving Instruments

If the aim of modelling an acoustic instrument is to reproduce its original operation, only in a digital form, a performer is usually provided with control features that replicate what is available with the physical object. Features attributed to properties of a model such as material or geometry, are not considered controllable during performance, as they would not be in a real instrument. However, when designing an infeasible instrument, which does not have a real counterpart, any model feature could be considered a performance parameter. Thus, one could play an instrument that 'shape-shifts' or 'transmutes' during performance in a controllable way. A performer might make an instrument evolve and exploit it for musical purposes.

5.6.1. Evolution Parameters

Due to wide variety of available or prospective models and their increasing complexity, a set of evolution control parameters is an open one. Some examples may be given on the basis of hitherto presented models of string and hyper-membrane, implemented in parallel using OpenCL. A complete list of model parameters for a string and a hyper-membrane is provided in Tables 5.2 and 5.3, respectively. However, only a subset of this list can be considered relevant choices. There are two conditions for a valid evolution parameter:

- while adjusted, it has to produce a perceptible auditory effect,
- its impact on the output signal has to be of a continuous nature.

While the first condition is met by most of the parameters, the second one eliminates a considerable amount of possibilities. Such is the case of hammer-related parameters which can only manifest their impact during short periods of string contact. Adjustments performed in between will affect the signal during next contact, thus the effect

of change is not an evolution, but a discontinuous series of distinct sound events. Clearly, the most useful parameters shall regard a vibrating element itself.

What is left in case of the string model is the string inharmonicity `B` related to stiffness parameter κ (3.132), string fundamental frequency `f0` related through γ (3.121) and c (3.111) to its tension, thickness and mass (3.109), and finally loss parameters σ_0 and σ_1 controlled through more intuitive `lf1`, `lf2`, `lT1`, and `lT2` (3.137). In case of the hyper-membrane a selection is very narrow, yet what remains, i.e. domain aspect ratios x_i, may prove particularly interesting, if not peculiar.

There remains an additional parameter that could be exploited if handled properly. Bilbao describes a technique allowing to introduce temporal variations to a signal produced by a string [61]. It involves continuous altering the location, i.e. the choice, of a readout grid point. Various output operators, such as (3.89) or (3.91), apply interpolation to make changes appear continuous. In case of a string the technique is aimed at simulating movements of a musician and an instrument in relation to a receiver, characteristic for bowed string instruments, particularly violin.

While the technique is a simple attempt to reproduce a real effect, it might be expanded to an infeasible case of hyper-membrane. It may be considered a hybrid of physical modelling and wave terrain method, only with terrain and orbit being not two-, but higher-dimensional. An N-dimensional grid function, itself evolving, can be regarded as a terrain, and a floating readout location – an orbit. Complex orbits, such as N-dimensional variants of examples presented in Figure 2.30 can be applied.

5.6.2. Means of Control

Once an evolution parameter has been selected, there are several possibilities to guide its variation:

- allow a user to adjust it directly through some form of a performance controller,
- apply a beforehand sampled function,
- use a function given by an expression,
- apply an envelope in a form of piecewise-linear approximation,
- control it by a low-frequency oscillator (LFO).

Variation through either one of two last positions may be considered a hybrid variant of a source-modifier method: based on a physical-modelling, yet employing subtractive-type modifiers. Like in the subtractive synthesis, a parameter is constantly modified by an envelope or LFO, while the role of a user is to control the modifier features, such as depth or frequency. Going further the way of source-modifier methods, one could attempt a modular approach to control and parameters, though it would require more complex models with larger sets of parameters. A selected evolution parameter could be related to a continuously-changing characteristic of a model or some output-derived value. However, considering simple models, such as presented strings and membranes, the first and two last options shall be of the greatest utility.

5.6.3. Implementation Consideration

Main concern regarding alteration of model parameters has been stated while discussing implementation of a string model: adjustment of a single parameter involves correcting other parameters as well. Even if it is not directly related, most cases lead to necessity of creating a new grid in order for a model to remain near the stability condition. State of a new grid has to closely match the previous one, if a sound is to continue, which is exactly the case of evolving instruments. This may be accomplished through interpolating between the old grid, and the new one.

Out of parameters mentioned as candidates for controlling model evolution, i.e. string inharmonicity, aspect ratios in hyper-membrane, and readout location, only the last one does not require calculation of a new grid. The second one involves it directly, and the first one, inharmonicity, indirectly, due to stability condition readjustment.

OpenCL implementations of both, a stiff string model and a hyper-membrane, relegate actual simulation from the host to the kernel, intended to run on a GPU. For the sake of efficiency, on a single run kernel calculates not a single time step, that would produce one signal sample only, but a series of samples filling an audio buffer. The refresh rate of a control data might be much below the sample rate – it is updated once per full buffer of signal samples. This might lead to audible discontinuities in produced signal.

The simplest, even though not the most efficient way to address the issue, is to decrease size of an audio buffer calculated in a single kernel run. This directly increases frequency of control data updates, and reduces differences between subsequent values of controlled parameter. A better solution may be applied if the parameter is controlled by envelope or LFO. In such case it is possible to calculate intermediate parameter values and send them to kernel as an array. Thus the kernel would use subsequent, gradually changing values during a single run. This method can also be combined with parameter controlled by a user directly. Instead of using one user-provided parameter value per the entire run of a kernel, the program can determine ratio of change from the previous to the new value, and produce a short envelope for the span of a single kernel run.

It is to be considered though, that in case of most evolution parameters each update requires a resize of the grid through N-dimensional interpolation. Not only it increases amount of calculations considerably, thus has an impact on model performance, but it also leads to signal degradation with each successive grid interpolation.

5.6.4. Selected Examples

5.6.4.1. Evolving Material Parameter

Evolution of material parameter has been applied to the model of a stiff string with a loss given by (5.1). Inharmonicity has been chosen as the evolution parameter. It varied within the range of [0.0001, 1.60001], and was controlled by 2 Hz sine LFO. Model parameters were set according to Table 5.2, with the following exceptions: `SR`=96 kHz, `vH0`=20, `f0`=110.5 Hz, `lf1`=1.3 kHz, and `lf2`=13 kHz. One-dimensional

interpolation was performed using non-rounded Akima spline with natural boundary conditions and non-rounded corner algorithm of Wodicka from the GNU Scientific Library [9].

The result of simulation is presented in Figure 5.21. It may be compared to the *wah-wah* effect due to periodically increasing signal bandwidth. However, inharmonicity evolution affects not only spectral envelope, but a spectral structure as well – increasing bandwidth is accompanied by detuning of higher partials. Therefore the auditory effect can be categorised not only as timbre-related, but as pitch-related as well.

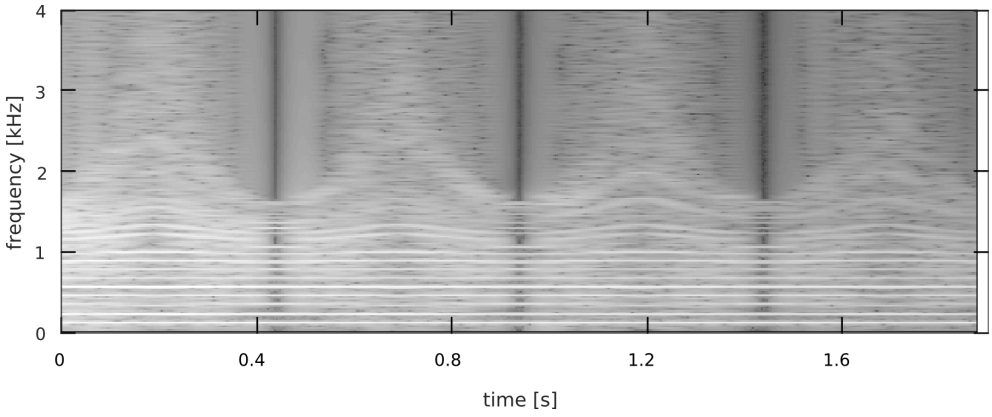


Figure 5.21. Spectrogram of a signal produced by evolving string model defined in (5.1), with stiffness parameter modulated using 2 Hz sine; details are provided in text

5.6.4.2. Evolving Shape

Evolution of shape is presented on the simplest case of a 2D membrane given by (5.9). 2D grid has a single aspect ratio and it has been used as the evolution parameter. Three signal variants have been produced. In two of them aspect ratio varied within the range of [1.0, 2.0], and in the remaining one the range has been increased to [1.0, 5.0]. Simulation parameters were set according to Table 5.4, with the exceptions of `SR`=44.1 kHz. In two cases the parameter was controlled by a simple linear envelope shifting from the larger to the smaller value, and in one it was controlled by 4 Hz sine LFO. Two-dimensional interpolation was performed using bicubic method [294].

The results of simulation are presented in Figure 5.22. A very interesting spectral behaviour may be observed in two cases of linear parameter adjustment. While adjusting the shape, the overall grid size is attempted to be maintained. Therefore, with two grid dimensions changing in opposite directions, some partials are shifted upwards, and the remaining ones – downwards. The effect bears some auditory resemblance to the Shepard–Risset *glissando* [465] occurring in both directions simultaneously. Sine-modulated parameter produces a 'spring-like' auditory effect with an ambiguous sensation of pitch direction.

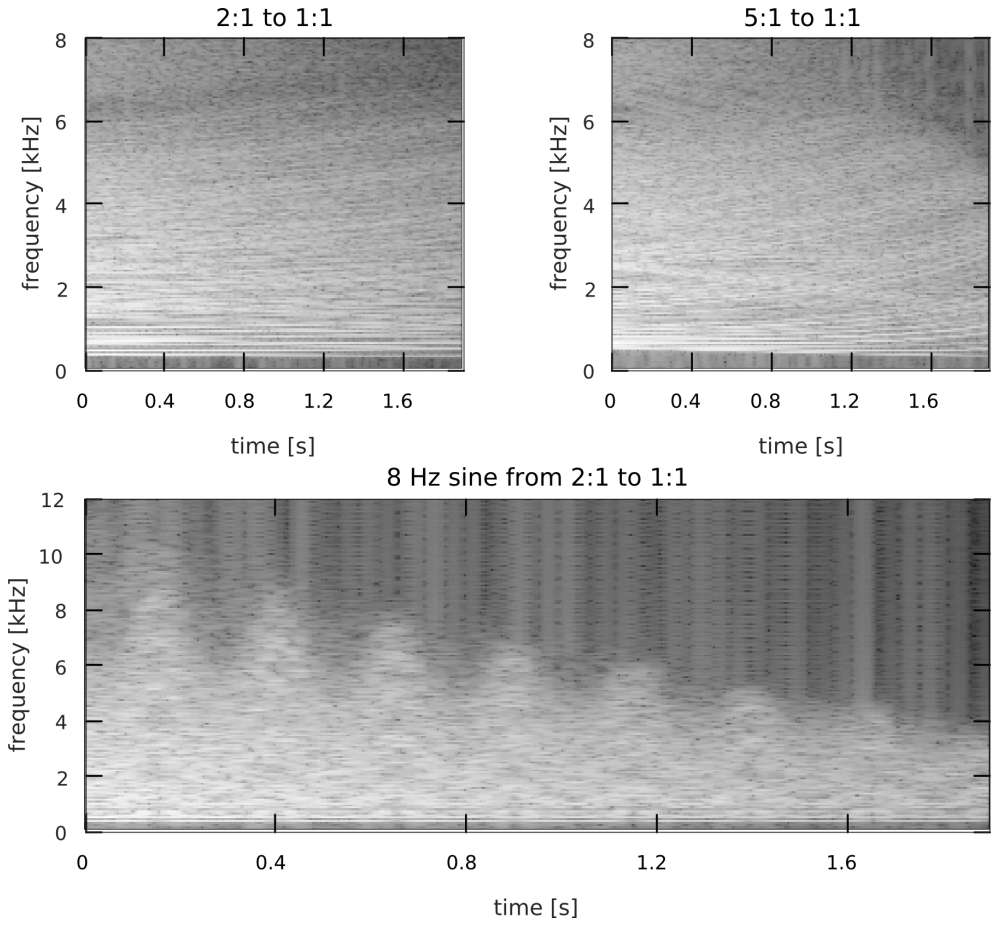


Figure 5.22. Spectra of signals produced by evolving 2D grids according to model (5.9); grid aspect ratio either changes linearly (top plots), or is modulated using 4 Hz sine; details are provided in text

5.6.4.3. Floating Readout

The third kind of instrument evolution, which is a hybrid of a quasi-physical model and a waveterrain method, has been presented using a 3D hyper-membrane given by (5.9) with output of the instrument obtained through a floating readout location. Due to three dimensions of a grid, a three-dimensional 'terrain' scanning orbit has been applied as well. A common choice of trajectory in case of 2D waveterrain synthesis

is the Lissajous curve, therefore its three-dimensional variant, also referred to as the Lissajous knot [73], has been used. It is given by the following expression

$$\begin{aligned} x_1 &= A_1 \sin(t) \\ x_2 &= A_2 \sin(n_2 t + \phi) \\ x_3 &= A_3 \sin(n_3 t + \psi) \end{aligned} \tag{5.23}$$

which can be easily generalised to arbitrary number of dimensions, thus it is well suited as an orbit for hyper-membrane terrains. Parameter A_i controls spatial extent of the figure in each dimension, while parameter n_i allows to adjust its spatial density. By adjusting size and adding offset to obtained coordinates it is possible to constrain scanning to a limited part of terrain only. Figure 5.23 presents Lissajous knots applied in the simulation.

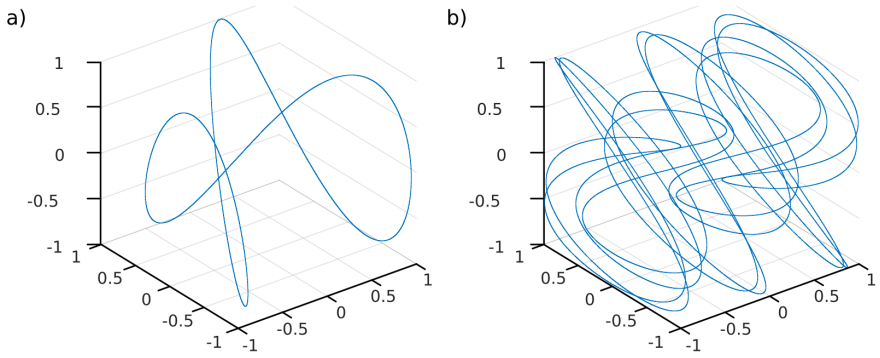


Figure 5.23. Lissajous knots, as defined in (5.23), used in simulation: a) $n_2 = 2$, $n_3 = 3$; b) $n_2 = 17$, $n_3 = 11$

Apart from floating readout location and sampling frequency set to 44.1 kHz, simulation parameters were set as in rectangular cuboid in Figure 5.14. In order to reduce discontinuities while changing the readout point, a trilinear interpolation operator, equivalent to a bilinear variant from (3.94), has been applied on the basis of eight grid points around exact observation location:

$$\begin{aligned} I_1(x_{1,o}, x_{2,o}, x_{3,o}) u_{(l_1,l_2,l_3)} = &(1-\alpha_{(x_1,o)})(1-\alpha_{(x_2,o)})(1-\alpha_{(x_3,o)}) u_{(l_{1,o},l_{2,o},l_{3,o})} + \\ &+ \alpha_{(x_1,o)}(1-\alpha_{(x_2,o)})(1-\alpha_{(x_3,o)}) u_{(l_{1,o}+1,l_{2,o},l_{3,o})} + \\ &+ (1-\alpha_{(x_1,o)})\alpha_{(x_2,o)}(1-\alpha_{(x_3,o)}) u_{(l_{1,o},l_{2,o}+1,l_{3,o})} + \\ &+ (1-\alpha_{(x_1,o)})(1-\alpha_{(x_2,o)})\alpha_{(x_3,o)} u_{(l_{1,o},l_{2,o},l_{3,o}+1)} + \\ &+ (1-\alpha_{(x_1,o)})\alpha_{(x_2,o)}\alpha_{(x_3,o)} u_{(l_{1,o},l_{2,o}+1,l_{3,o}+1)} + \\ &+ \alpha_{(x_1,o)}(1-\alpha_{(x_2,o)})\alpha_{(x_3,o)} u_{(l_{1,o}+1,l_{2,o},l_{3,o}+1)} + \\ &+ \alpha_{(x_1,o)}\alpha_{(x_2,o)}(1-\alpha_{(x_3,o)}) u_{(l_{1,o}+1,l_{2,o}+1,l_{3,o})} + \\ &+ \alpha_{(x_1,o)}\alpha_{(x_2,o)}\alpha_{(x_3,o)}) u_{(l_{1,o}+1,l_{2,o}+1,l_{3,o}+1)} \end{aligned} \tag{5.24}$$

where $(x_{1,o}, x_{2,o}, x_{3,o})$ are the observation coordinates, $u_{(l_1,l_2,l_3)}$ is the grid function, $(l_{1,o}, l_{2,o}, l_{3,o})$ are the coordinates of a grid point obtained by truncation of the observation coordinates, and $\alpha_{(x_i,o)}$ is the fractional part of the observation point along i-th coordinate. Expression (5.24) can be easily extended to arbitrary number of dimensions, although number of grid points required for approximation will grow exponentially.

Three cases of readout evolution have been compared to a fixed readout location. They are presented in Figure 5.24. Depending on location of observation point, amplitudes of different signal partials are reduced. Therefore a floating readout location has an effect of a filter-bank with time-varying parameters.

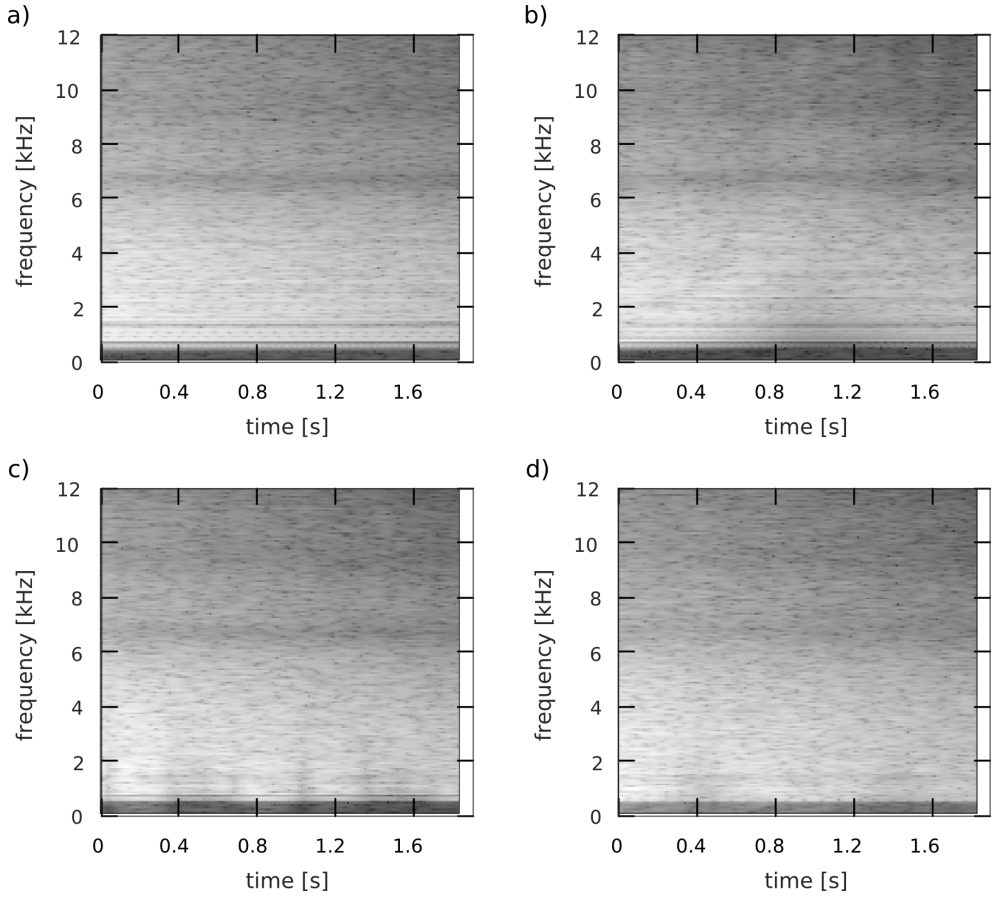


Figure 5.24. Signals produced by a 3D hyper-membrane with readout location floating along Lissajous knot: a) fixed location; b) $\frac{1}{10}$ of figure period per full signal duration with $n_2 = 2$, $n_3 = 3$; c) full period per signal duration with $n_2 = 2$, $n_3 = 3$; d) full period with $n_2 = 17$, $n_3 = 11$

Auditory effect strongly depends on the frequency of terrain scanning. Slow changes (Fig. 5.24b) produce gradually evolving spectra. Moderate changes introduce a temporal structure of a rhythmic character. Finally, fast changes have a character of tremolo-like amplitude modulation with varying brightness, or a spectro-temporal texture. The important features of the effect are convenient control over its scale by adjusting a few orbit parameters, and its diverse character oscillating between timbre, rhythm and pitch sensation.

5.6.5. Further Study

Clearly, three types of instrument evolution presented, i.e. its shape, material, or orientation towards observation point, do not exhaust possible scenarios of instruments that change over the course of produced sound. In cases of bowed strings or wind instrument models it might be worth to explore evolution of excitation controlled through parameters normally fixed during a performance. In another scenario chosen simulation principle could be gradually altered. For instance, one could apply boundary conditions that have controllable parameters. Such is the case of Robin conditions [225], sometimes referred to as impedance boundary conditions, which are weighted linear combination of a Dirichlet and Neumann type.

Scenarios presented are far from being completely studied as well. Even in models as simple as hitherto discussed, some material parameters remain to be studied, e.g. loss-related ones. More complex, realistic models would provide new parameters. Evolution of shape or geometry may be applied to boundary loops, e.g. with looped edges shifted in relation to each other. Alternatively, one might explore the effect of evolving *boundary mask*. Floating observation point may employ various kinds of orbits apart from presented Lissajous knots, not only given by arithmetic formula, but e.g. sampled or otherwise user-defined.

5.7. Concluding Remarks

A concept of infeasible instruments is aimed at development of sound synthesis techniques for real-time performance that combine simple, meaningful control facilities and sound production mechanisms of acoustic instruments with synthetic abilities to design and create new sounds and expressive effects. While the concept is open, an approach based on finite difference method has been discussed in more details. As one of physical modelling methods, FD schemes can approximate behaviour of real instruments or predict behaviour of instruments that, due to various reasons, cannot exist. Some of these infeasible objects can produce complex signals, containing various detailed structures that are controlled through a relatively small parameter set that can be intuitively mapped onto an instrument-like musical controller.

Three cases of instrument infeasibility has been proposed an studied. The first one is based on increasing dimensionality of vibrating elements, which combined with artifacts of numerical approximation leads to emergent behaviour and production of

detailed spectro-temporal structures, efficiently controlled with conveniently limited set of parameters. The second case alters operation of domain boundaries through setting boundary loops. Finally, the third case involves performance-time alteration and control over parameters of simulation that could not be controlled by a performer in acoustic instruments. Each of these cases produces specific auditory effect that may be considered as a building block and combined with others to 'design' a sound possessing desirable qualities.

Physical modelling of musical instruments may require a considerable amount of processing power in order to achieve real-time synthesis capabilities. Therefore an approach based on a hybrid, parallel implementation of FD schemes using CPUs and GPUs through the OpenCL framework, has been presented. Some of more complex, particularly higher-dimensional models may be beyond capabilities of current PC hardware, yet with recent progress aimed primarily at parallel processing, the limit of what is possible to simulate in real time is pushed away.

Promising effects of introductory simulations and improving facilities for implementation of complex physical models allows to initiate a broader research towards design of infeasible instruments, as well as their direct musical applications.

6. Conclusions of the Monograph

The objective of the monograph was to present current state of sound synthesis methods as well as to highlight the author's contribution into the filed of sound synthesis, in the form of two new synthesis methods: **phrase assembling synthesis**, and **infeasible instruments**. The most important results are summarised below.

In the first chapter distinctive features of sound synthesis have been discussed in comparison to traditional musical instruments. A new definition of sound synthesis have been formulated. A taxonomy of synthesis methods has been proposed, with a new division into direct and indirect methods, using a criterion of presence of an intermediate model or idea between the control parameters and parameters of produced signal.

In the second chapter a survey of direct methods of sound synthesis has been carried out, including spectral and waveform based techniques. Presented methods include additive, subtractive, wavetable, sampling, granular, and concatenative synthesis. New developments in additive synthesis have been presented. Concatenative synthesis has been given a detailed review, due to lack of such review in to date literature concerning sound synthesis in general, and due to features shared with author's method of phrase assembling synthesis.

In the third chapter indirect methods of sound synthesis have been presented. Discussed methods include abstract and physical modelling methods. The former include frequency modulation, waveshaping, and a collection of non-standard synthesis techniques. The latter consist of finite difference approximations, networks of lumped elements, modal synthesis, Karplus–Strong and waveguide synthesis, as well as a set of various emerging techniques. Finite difference approximations have been given a more thorough review, with many ready to use approximation schemes, as a foundation of infeasible instruments.

The fourth chapter concerns the first of two methods proposed by the author: the phrase assembling synthesis. In the initial part of the chapter two synthesis methods commonly applied to score reproduction purposes have been presented. Their features have been compared, and their issues analysed. The issues allowed to formulate assumptions regarding a new method, aiming at solving selected problems. The most important of indicated issues are the inability of sampling synthesis to reproduce natural note transitions, management and usage of large collections of samples, as

well as complexity and dependence on external tools on the part of the concatenative synthesis. The concept of the author's proposition of solving or attenuating these issues has been presented. It relies on recording a large collection of multi-pitch samples that represent all pitch transitions possible in a given instrument reproducing an orchestral part. The samples are connected into phrases on common notes using phase aligned crossfade. Samples contain low level expressive musical features. High level expressive features are imposed on the output signal through the application of performance rules. Selection of samples and rules is based on prior automatic score analysis. All the designed algorithms and techniques regarding assembling of phrases have been presented and discussed in detail. The method has been implemented as a set of GNU Octave scripts, and tested in order to adjust algorithms parameters. Test results have been provided in the final sections of the chapter.

The second method proposed by the author, the infeasible instruments synthesis, has been presented in the fifth chapter. The chapter has stated the requirements for real-time performance methods with flexible timbre control capabilities. Next, the concept of infeasible instrument has been formulated: it is a model of musical instrument, or a sound producing object in general, that cannot be built in real world due to various reasons. Such instrument shall retain enough of real object's characteristics in order to be controlled in a meaningful, intuitive manner by a performer. At the same time, introduced modifications shall lead to production of new, distinctive, and musically appealing sounds. Operation of infeasible instruments involves performing a finite difference simulation. Such simulations may be computationally demanding, therefore the author has presented a detailed discussion regarding the implementation of the method on the graphics processing units, which allows to design more complex models in comparison to calculations performed on the CPU only. The program has been implemented in the C programming language with the OpenCL framework for the GPU computing. It has been supplemented with a networked controller program implemented in PureData environment. Several infeasible instruments have been presented in subsequent sections, including hyper-membranes, membranes with various periodic boundaries, as well as instruments having selected properties changing over time. Features of all models have been discussed, with regards to certain musical characteristics. The concept is open, and more models, applying different principles, may be designed on the basis of presented framework.

Both methods have been presented in the context of current synthesis methods. Their features, advantages, and issues have been discussed. The scope of details presented, with either algorithms provided in the form of flow charts, or selected sections of source code presented and discussed, should allow to replicate and expand the designs by interested researchers or sound engineers. Thus the author hopes to contribute to the field of sound synthesis, and sound engineering in general.

Bibliography

[1] J. S. Abel, V. Välimäki, and J. O. Smith III. Robust, Efficient Design of Allpass Filters for Dispersive String Sound Synthesis. *IEEE Signal Processing Letters*, 17(4):406–409, Apr. 2010.

[2] S. Adachi and M. Sato. Time-domain simulation of sound production in the brass instrument. *The Journal of the Acoustical Society of America*, 97(6):3850–3861, June 1995.

[3] D. Adamczyk and I. Czajka. Modelowanie sprzężenia pola akustycznego i pola przepływu na przykładzie fletu ze zbiorów Muzeum Śląska Cieszyńskiego. In *Proceedings of the XXII Conference on Acoustics and Biomedical Engineering*, pages 112–119, 2018.

[4] N. H. Adams, M. A. Bartsch, and G. H. Wakefield. Note segmentation and quantization for music information retrieval. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(1):131–141, 2006.

[5] A. B. Adib. Study Notes on Numerical Solutions of the Wave Equation with the Finite Difference Method. *ArXiv*, 2000.

[6] J. M. Adrien. *Representations of Musical Signals*, chapter The missing link: modal synthesis, pages 269–297. MIT Press, 1991.

[7] J. M. Adrien and E. Ducasse. Dynamic Modeling of Vibrating Structures for Sound Synthesis, Modal Synthesis. In *Proceedings of the Audio Engineering Society Conference: 7th International Conference: Audio in Digital Times*, pages 291–299, May 1989.

[8] S. Aisyah. FPGA-based sound synthesis by digital waveguide. In *Proceedings of 6th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO)*. IEEE, May 2015.

[9] H. Akima. A Method of Bivariate Interpolation and Smooth Surface Fitting for Irregularly Distributed Data Points. *ACM Transactions on Mathematical Software*, 4(2):148–159, June 1978.

[10] A. Almeida, R. Chow, J. Smith, and J. Wolfe. The kinetics and acoustics of fingering and note transitions on the flute. *The Journal of the Acoustical Society of America*, 126(3):1521–1529, Sept. 2009.

[11] A. Almeida, C. Vergez, and R. Caussé. Quasistatic nonlinear characteristics of double-reed instruments. *The Journal of the Acoustical Society of America*, 121(1):536–546, Jan. 2007.

[12] A. Almeida, C. Vergez, R. Caussé, and X. Rodet. Physical study of double-reed instruments for application to sound synthesis. In *Proceedings of the International Symposium on Musical Acoustics, Cancun, Mexico*, pages 221–226, 2002.

[13] P. Alonso, R. Cortina, F. J. Rodríguez-Serrano, P. Vera-Candeas, M. Alonso-González, and J. Ranilla. Parallel online time warping for real-time audio-to-score alignment in multi-core systems. *The Journal of Supercomputing*, 73(1):126–138, 2017.

[14] X. Amatriain, J. Bonada, A. Loscos, J. L. Arcos, and V. Verfaille. Content-based Transformations. *Journal of New Music Research*, 32(1):95–114, 2003.

[15] C. Ames. Automated Composition in Retrospect: 1956–1986. *Leonardo*, 20(2):169–185, 1987.

[16] C. Ames. The Markov Process as a Compositional Model: A Survey and Tutorial. *Leonardo*, 22(2):175–187, 1989.

[17] G. V. Anand. Large-Amplitude Damped Free Vibration of a Stretched String. *The Journal of the Acoustical Society of America*, 45(5):1089–1096, May 1969.

[18] D. Arfib. Digital Synthesis of Complex Spectra by Means of Multiplication of Nonlinear Distorted Sine Waves. *Journal of the Audio Engineering Society*, 27(10):757–768, 1979.

[19] D. Arfib. *Representations of Musical Signals*, chapter Analysis, transformation, and resynthesis of musical sounds with the help of a time-frequency representation, pages 87–118. The MIT Press, 1991.

[20] A. Arzt. Score Following with Dynamic Time Warping. Master's thesis, Vienna University of Technology, 2007.

[21] B. Atal and J. Remde. A new model of LPC excitation for producing natural-sounding speech at low bit rates. In *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing*, pages 614–617, 1982.

[22] M. Atig, J.-P. Dalmont, and J. Gilbert. Termination impedance of open-ended cylindrical tubes at high sound pressure level. *Comptes Rendus Mécanique*, 332(4):299–304, Apr. 2004.

[23] J.-J. Aucouturier and F. Pachet. Ringomatic: A Real-Time Interactive Drummer Using Constraint-Satisfaction and Drum Sound Descriptors. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR), London*, pages 412–419, 2005.

[24] J.-J. Aucouturier, F. Pachet, and P. Hanappe. From sound sampling to song sampling. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 1–8, 2004.

[25] F. Avanzini and D. Rocchesso. Efficiency, accuracy, and stability issues in discrete-time simulations of single reed wind instruments. *The Journal of the Acoustical Society of America*, 111(5):2293, 2002.

[26] F. Avanzini and M. van Walstijn. Modelling the Mechanical Response of the Reed-mouthpiece-lip System of a Clarinet. Part I. A One-Dimensional Distributed Model. *Acta Acustica united with Acustica*, 90(3):537–547, May 2004.

[27] J. Backus. *The Acoustical Foundations of Music*. Norton, New York, second edition, 1977.

[28] J. Backus. Multiphonic tones in the woodwind instruments. *The Journal of the Acoustical Society of America*, 63(2):591–599, Feb. 1978.

[29] R. A. Bacon and J. M. Bowsher. A Discrete Model of a Struck String. *Acta Acustica united with Acustica*, 41(1):21–27, Oct. 1978.

[30] R. Bader. *Computational Mechanics of the Classical Guitar*. Springer, Berlin–Heidelberg, 2005.

[31] P.-F. Baisnèe, J.-B. Barrière, M.-A. Dalbavie, J. Duthen, M. Lindberg, Y. Potard, and K. Saariaho. ESQUISSE: A Compositional Environment. In *Proceedings of the 1988 International Computer Music Conference, San Francisco*, pages 108–118, 1988.

[32] B. Bank. *Physics-based Sound Synthesis of String Instruments Including Geometric Nonlinearities*. PhD thesis, Budapest University of Technology and Economics, 2006.

[33] B. Bank and L. Sujbert. Generation of longitudinal vibrations in piano strings: From physics to sound synthesis. *The Journal of the Acoustical Society of America*, 117(4):2268–2278, Apr. 2005.

[34] A. Barjau, V. Gibiat, and N. Grand. Study of woodwind-like systems through nonlinear differential equations. Part I. Simple geometry. *The Journal of the Acoustical Society of America*, 102(5):3023–3031, Nov. 1997.

[35] A. Barjau, D. H. Keefe, and S. Cardona. Time-domain simulation of acoustical waveguides with arbitrarily spaced discontinuities. *The Journal of the Acoustical Society of America*, 105(3):1951–1964, Mar. 1999.

[36] J.-B. Barrière, Y. Potard, and P.-F. Baisnèe. Models of continuity between synthesis and processing for the elaboration and control of timbre structure. In B. Truax, editor, *Proceedings of the 1985 International Computer Music Conference, San Francisco*, pages 193–198, 1985.

[37] S. C. Bass and T. W. Goeddel. The Efficient Digital Implementation of Subtractive Music Synthesis. *IEEE Micro*, 1(3):24–37, 1981.

[38] M. Bastiaans. Gabor's expansion of a signal into Gaussian elementary signals. *Proceedings of the IEEE*, 68(4):538–539, 1980.

[39] M. Bastiaans. On the sliding-window representation of signals. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 33(4):868–873, 1985.

[40] M. Bastiaans and M. Geilen. On the discrete Gabor transform and the discrete Zak transform. *Signal Processing*, 49(3):151–166, 1996.

[41] J. A. Bate. The Effect of Modulator Phase on Timbres in FM Synthesis. *Computer Music Journal*, 14(3):38–45, 1990.

[42] E. Bavu, J. Smith, and J. Wolfe. Torsional Waves in a Bowed String. *Acta Acustica united with Acustica*, 91(2):241–246, 2005.

[43] J. Beauchamp. Brass-Tone Synthesis by Spectrum Evolution Matching with Nonlinear Functions. *Computer Music Journal*, 3(2):35–43, 1979.

[44] J. Beauchamp and A. Horner. Extended Nonlinear Waveshaping Analysis/Synthesis Technique. In *Proceedings of the International Computer Music Conference, San Francisco*, 1992.

[45] A. H. Benade. Mathematical Theory of Woodwind Fingerholes. *The Journal of the Acoustical Society of America*, 31(11):1564–1564, Nov. 1959.

[46] A. H. Benade. On the Mathematical Theory of Woodwind Finger Holes. *The Journal of the Acoustical Society of America*, 32(12):1591–1608, Dec. 1960.

[47] A. H. Benade. *Fundamentals of Musical Acoustics*. Dover Publications, 1990.

[48] A. H. Benade and E. V. Jansson. On Plane and Spherical Waves in Horns with Nonuniform Flare: I. Theory of Radiation, Resonance Frequencies, and Mode Conversion. *Acustica*, 31(2):79–98, 1974.

[49] G. Bennett and X. Rodet. *Current Directions in Computer Music Research*, chapter Synthesis of the singing voice, pages 19–44. MIT Press, Cambridge, MA, 1989.

[50] J. Bensa, S. Bilbao, R. Kronland-Martinet, and J. O. Smith III. The simulation of piano string vibration: From physical models to finite difference schemes and digital waveguides. *The Journal of the Acoustical Society of America*, 114(2):1095–1107, Aug. 2003.

[51] D. Benson. *Music: A Mathematical Offering*. Cambridge University Press, 2006.

[52] P. Berg. *A user's manual for SSP*. Utrecht: Institute of Sonology.

[53] P. Berg. PILE: A Language for Sound Synthesis. *Computer Music Journal*, 3(1):30–41, 1979.

[54] H. M. Berger. A New Approach to the Analysis of Large Deflections of Plates. *Journal of Applied Mathematics*, 22:465–472, 1955.

[55] E. R. Berlekamp, J. H. Conway, and R. K. Guy. *Winning Ways for your Mathematical Plays*. Academic Press, 1982.

[56] G. Bernardes, C. Guedes, and B. Pennycook. EarGram: an Application for Interactive Exploration of Large Databases of Audio Snippets for Creative Purposes. In *Proceedings of the 9th International Symposium on Computer Music Modeling and Retrieval (CMMR), London, UK*, pages 265–277, 2012.

[57] A. D. Bernstein and E. D. Cooper. The Piecewise-Linear Technique of Electronic Music Synthesis. *Journal of the Audio Engineering Society*, 24(6):446–454, 1976.

[58] P. Beyls. The Musical Universe of Cellular Automata. In *Proceedings of the International Computer Music Conference (ICMC), Columbus*, pages 34–41, 1989.

[59] S. Bilbao. Parameterized families of finite difference schemes for the wave equation. *Numerical Methods for Partial Differential Equations*, 20(3):463–480, 2004.

[60] S. Bilbao. Direct simulation for wind instrument synthesis. In *Proceedings of the 11th International Digital Audio Effects Conference, Espoo, Finland*, 2008.

[61] S. Bilbao. *Numerical Sound Synthesis: Finite Difference Schemes and Simulation in Musical Acoustics*. John Wiley & Sons, 2009.

[62] S. Bilbao and J. O. Smith III. Energy-conserving Finite Difference Schemes for Nonlinear Strings. *Acta Acustica united with Acustica*, 91(2):299–311, Apr. 2005.

[63] S. Bilbao, A. Torin, P. Graham, J. Perry, and G. Delap. Modular Physical Modeling Synthesis Environments on GPU. In *Proceedings of the International Computer Music Conference*, Sept. 2014.

[64] J. Bischoff, R. Gold, and J. Horton. A microcomputer-based network for live performance. *Computer Music Journal*, 2(3):24–29, 1978.

[65] I. Bisnovatyi. Flexible Software Framework for Modal Synthesis. In *Proceedings of the 3rd International Digital Audio Effects Conference*, pages 109–114, 2000.

[66] H. S. Black. *Modulation Theory*. Van Nostrand, New York–London, 1953.

[67] E. D. Blackham. The physics of the piano. *Scientific American*, pages 88–99, Dec. 1965.

[68] J. Blackledge, G. Evans, and P. Yardley. *Numerical Methods for Partial Differential Equations*. Springer, London, 1999.

[69] H. Blockeel, L. De Raedt, and J. Ramon. Top-down Induction of Clustering Trees. In *Proceedings of the 15th International Conference on Machine Learning, San Francisco*, pages 55–63, 1998.

[70] M. Blok. On Sample Rate Conversion Based on Variable Fractional Delay Filters. *International Journal of Computer Science and Applications*, 10(1):98–116, 2013.

[71] M. Blok and P. Drózda. Variable Ratio Sample Rate Conversion Based on Fractional Delay Filter. *Archives of Acoustics*, 39(2):231–242, 2014.

[72] T. Blum. Review of Herbert Brün: SAWDUST. *Computer Music Journal*, 3(1):6–7, 1979.

[73] M. G. V. Bogle, J. E. Hearst, V. F. R. Jones, and L. Stoilov. Lissajous Knots. *Journal of Knot Theory and Its Ramifications*, 03(02):121–140, June 1994.

[74] J. Bonada and A. Loscos. Sample-Based Singing Voice Synthesizer By Spectral Concatenation. In *Proceedings of the Stockholm Music Acoustics Conference*, 2003.

[75] J. Bonada and X. Serra. Synthesis of the Singing Voice by Performance Sampling and Spectral Models. *IEEE Signal Processing Magazine*, 24(2):67–79, 2007.

[76] A. Borgonovo and G. Haus. Sound Synthesis by means of Two-Variable Functions: experimental criteria and results. *Computer Music Journal*, 10(4):57–71, 1986.

[77] P. Boulez. *Stocktakings from an Apprenticeship*. Oxford: Clarendon Press, 1991.

[78] P. Bowcott. Cellular automata as a means of high level compositional control of granular synthesis. In *Proceedings of the 1989 International Computer Music Conference, San Francisco*, 1989.

[79] E. Brandt. Hard Sync Without Aliasing. In *Proceedings of International Computer Music Conference, Havana*, pages 365–368, 2001.

[80] C. A. Brebbia and R. D. Ciskowski, editors. *Boundary Element Methods in Acoustics*. Springer, Netherlands, 1991.

[81] M. Brend. *Strange Sounds: Offbeat Instruments and Sonic Experiments in Pop*. Backbeat, 2005.

[82] R. Bresin. Artificial neural networks based models for automatic performance of musical scores. *Journal of New Music Research*, 27(3):239–270, Sept. 1998.

[83] R. Bresin, G. De Poli, and R. Ghetta. Fuzzy performance rules. In A. Friberg and J. Sundberg, editors, *Proceedings of the KTH Symposium on Grammars for Music Performance*, pages 15–36, 1995.

[84] R. Bresin, A. Friberg, and J. Sundberg. Director musices: The KTH performance rules system. In *Proceedings of SIGMUS-46, Kyoto*, pages 43–48, 2002.

[85] R. Bristow-Johnson. Wavetable Synthesis 101, A Fundamental Perspective. In *Audio Engineering Society Convention 101*, Nov. 1996.

[86] P. Brossier, J. P. Bello, and M. D. Plumbley. Real-time temporal segmentation of note objects in music signals. In *Proceedings of the International Computer Music Conference (ICMC)*, 2004.

[87] C. Bruyns. Modal Synthesis for Arbitrarily Shaped Objects. *Computer Music Journal*, 30(3):22–37, Sept. 2006.

[88] D. Burraston and E. Edmonds. Cellular Automata in Generative Electronic Music and Sonic Art: Historical and Technical Review. *Digital Creativity*, 16(3):165–185, 2005.

[89] J. J. Burred. A framework for music analysis/resynthesis based on matrix factorization. In *Proceedings of the International Computer Music Conference (ICMC), Athens*, 2014.

[90] J. J. Burred, C.-E. Cella, G. Peeters, A. Röbel, and D. Schwarz. Using the SDIF Sound Description Interchange Format for Audio Features. In *Proceeding of the International Conference on Music Information Retrieval (ISMIR)*, pages 427–432, 2008.

[91] W. Buxton, S. Patel, W. Reeves, and R. Baecker. Object and the design of timbral resources. *Computer Music Journal*, 6(2):32–44, 1982.

[92] E. Bynum and T. D. Rossing. *The Science of String Instruments*, chapter Cello, pages 245–257. Springer, 2010.

[93] A. Cabboi and J. Woodhouse. Validation of a Constitutive Law for Friction-induced Vibration under Different Wear Conditions. *Wear*, 396–397:107–125, Feb. 2018.

[94] R. F. Cádiz and J. Ramos. Sound Synthesis of a Gaussian Quantum Particle in an Infinite Square Well. *Computer Music Journal*, 38(4):53–67, Dec. 2014.

[95] C. Cadoz, A. Luciani, and J. L. Florens. CORDIS-ANIMA: A Modeling and Simulation System for Sound and Image Synthesis: The General Formalism. *Computer Music Journal*, 17(1):19–29, 1993.

[96] C. Cadoz, A. Luciani, J. L. Florens, C. Roads, and F. Chadabe. Responsive Input Devices and Sound Synthesis by Stimulation of Instrumental Mechanisms: The CORDIS System. *Computer Music Journal*, 8(3):60–73, 1983.

[97] T. Cahill. Art of and apparatus for generating and distributing music electrically, 1897.

[98] R. Cann. *Speech analysis / synthesis for electronic vocal music*. PhD thesis, Princeton University Department of Music, 1978.

[99] J. J. Carabias-Orti, F. J. Rodríguez-Serrano, P. Vera-Candeas, N. Ruiz-Reyes, and F. J. Canadas-Quesada. An Audio to Score Alignment Framework Using Spectral Factorization and Dynamic Time Warping. In *Proceedings of the 16th International Society for Music Information Retrieval (ISMIR) Conference*, pages 742–748, 2015.

[100] M. Cardle, S. Brooks, Z. Bar-Joseph, and P. Robinson. Sound-by-Numbers: Motion-Driven Sound Synthesis. In *Proceedings of the SIGGRAPH Symposium on Computer Animation*, 2003.

[101] G. F. Carrier. On the nonlinear vibration problem of the elastic string. *Quarterly of Applied Mathematics*, 3:157–165, 1945.

[102] A. A. Carrillo. *Gesture based synthesis of bowed string instruments*. PhD thesis, Universitat Pompeu Fabra, 2006.

[103] J. R. Carson. Notes on the Theory of Modulation. *Proceedings of the Institute of Radio Engineers*, 10(1):57–64, 1922.

[104] M. A. Casey. Acoustic Lexemes for Organizing Internet Audio. *Contemporary Music Review*, 24(6):489–508, Apr. 2005.

[105] M. A. Casey, R. Veltkamp, M. Goto, M. Leman, C. Rhodes, and M. Slaney. Content-Based Music Information Retrieval: Current Directions and Future Challenges. In *Proceedings of the IEEE*, volume 96, pages 668–696. Institute of Electrical and Electronics Engineers (IEEE), Apr. 2008.

[106] D. M. Causon and C. G. Mingham. *Introductory Finite Difference Methods for PDEs*. Ventus Publishing ApS, 2010.

[107] S. Cavaliere and A. Piccialli. *Musical Signal Processing*, chapter Granular synthesis of musical signals, pages 155–186. Lisse: Swets & Zeitlinger, 1997.

[108] A. Chaigne and V. Doutaut. Numerical simulations of xylophones. I. Time-domain modeling of the vibrating bars. *The Journal of the Acoustical Society of America*, 101(1):539–557, Jan. 1997.

[109] A. Chaigne and C. Lambourg. Time-domain simulation of damped impacted plates. I. Theory and experiments. *The Journal of the Acoustical Society of America*, 109(4):1422–1432, Apr. 2001.

[110] D. Chakraborty and N. Kovvali. Generalized normal window for digital signal processing. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, May 2013.

[111] A. Chandra. The linear change of waveform segments causing non-linear changes of timbral presence. *Contemporary Music Review*, 10(2):157–169, 1994.

[112] J. Chareyron. Digital Synthesis of Self-Modifying Waveforms by Means of Linear Automata. *Computer Music Journal*, 14(4):25–41, 1990.

[113] S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic Decomposition by Basis Pursuit. *SIAM Journal on Scientific Computing*, 43(1):129–159, 2001.

[114] C.-S. Chien, S.-L. Chang, and Z. Mei. Tracing the buckling of a rectangular plate with the Block GMRES method. *Journal of Computational and Applied Mathematics*, 136(1-2):199–218, Nov. 2001.

[115] J. Chomiński. Technika sonorystyczna jako przedmiot systematycznego szkolenia [Sonoristic Technique as the Subject of a Systematic Training]. *Muzyka*, 6(3):3–10, 1961.

[116] J. M. Chowning. The Synthesis of Complex Audio Spectra by Means of Frequency Modulation. *Journal of the Audio Engineering Society*, 21(7):526–534, 1973.

[117] J. M. Chowning. Method of synthesizing a musical sound, 1977.

[118] J. M. Chowning. *Sound Generation in Winds, Strings, Computers*, chapter Computer synthesis of the singing voice, pages 4–13. Royal Swedish Academy of Music, Stockholm, 1980.

[119] J. M. Chowning. *Current Directions in Computer Music Research*, chapter Frequency modulation synthesis of the singing voice, pages 57–63. The MIT Press, 1989.

[120] E. F. Codd. *Cellular Automata*. Academic Press, London, 1968.

[121] P. Codognet and D. Diaz. Yet Another Local Search Method for Constraint Solving. In *Proceedings of the International Symposium on Stochastic Algorithms*, pages 73–90, 2002.

[122] G. Coleman. Mused: Navigating the personal sample library. In *Proceedings of the International Computer Music Conference (ICMC)*, 2007.

[123] G. Coleman. *Descriptor Control of Sound Transformations and Mosaicing Synthesis*. PhD thesis, Universitat Pompeu Fabra, Barcelona, Feb. 2016.

[124] G. Coleman, J. Bonada, and E. Maestre. Adding Dynamic Smoothing to Mixture Mosaicing Synthesis. In *Proceedings of Spars11: Workshop on Signal Processing with Adaptive Sparse Structured Representations*, Edinburgh, Scotland, UK, June 2011.

[125] G. Coleman, E. Maestre, and J. Bonada. Augmenting Sound Mosaicing with Descriptor-driven Transformation. In *Proceedings of the 13th International Conference on Digital Audio Effects (DAFx-10), Graz*, 2010.

[126] N. Collins and B. L. Sturm. Sound cross-synthesis and morphing using dictionary-based methods. In *Proceedings of the International Computer Music Conference (ICMC), Huddersfield*, pages 595–601, 2011.

[127] J. W. Coltman. Time-domain simulation of the flute. *The Journal of the Acoustical Society of America*, 92(1):69–73, July 1992.

[128] J. M. Comajuncosas. *The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing, and Programming*, chapter Wave Terrain Synthesis with CSound. MIT Press, 2000.

[129] H. A. Conklin. Generation of partials due to nonlinear mixing in a stringed instrument. *The Journal of the Acoustical Society of America*, 105(1):536–545, Jan. 1999.

[130] A. Cont, S. Dubnov, and G. Assayag. GUIDAGE: A Fast Audio Query Guided Assemblage. In *Proceedings of the International Computer Music Conference (ICMC), Copenhagen*, 2007.

[131] P. R. Cook. *Identification of Control Parameters in an Articulatory Vocal Tract Model with Applications to the Synthesis of Singing.* PhD thesis, Stanford University, 1990.

[132] P. R. Cook. TBone: An Interactive WaveGuide Brass Instrument Synthesis Workbench for the NeXT Machine. In *Proceedings of the International Computer Music Conference, Montreal, Canada*, pages 297–299, 1991.

[133] P. R. Cook. SPASM, a real-time vocal tract physical model controller; and a singer, the companion software synthesis system. *Computer Music Journal*, 17(1):30–44, 1993.

[134] P. R. Cook. Physically informed sonic modeling (PhISM): percussive synthesis. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 228–231, 1996.

[135] P. R. Cook. Physically informed sonic modeling (PhISM): Synthesis of percussive sounds. *Computer Music Journal*, 21(3):38–49, 1997.

[136] P. R. Cook. *Real Sound Synthesis for Interactive Applications.* A.K. Peters, 2002.

[137] R. D. Cook, D. S. Malkus, and M. E. Plesha. *Concepts and Applications of Finite Element Analysis.* John Wiley and Sons, New York, fourth edition, 2002.

[138] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematical Computation*, 19:297–301, 1965.

[139] M. Costa, P. Fine, P. Enrico, and R. Bitti. Interval Distributions, Mode, and Tonal Strength of Melodies as Predictors of Perceived Emotion. *Music Perception: An Interdisciplinary Journal*, 22(1):1–14, Sept. 2004.

[140] R. Courant, K. Friedrichs, and H. Lewy. Über die partiellen Differenzengleichungen der mathematischen Physik. *Mathematische Annalen*, 100:32–74, 1928.

[141] H. S. M. Coxeter. *Regular Polytopes*. Dover Publications, 1973.

[142] A. G. Crespi. Spectral Manipulation of Audio using General-Purpose Graphics Processing Units. Master's thesis, Politecnico di Milano, 2016.

[143] I. Czajka and A. Gołaś. *Inżynierskie metody analizy numerycznej i planowanie eksperymentu [Engineering methods of numerical analysis and experiment planning]*. Wydawnictwa AGH, 2017.

[144] A. Czerwiński, D. Grzybek, P. Krauze, J. Łuczko, K. Michalczyk, P. Orkisz, M. Pluta, L. Radziszewski, M. Saga, and J. Snamina. *Wybrane zagadnienia układów redukcji drgań i hałasu*, chapter Synteza dźwięku z wykorzystaniem procesora graficznego, sterowana przy użyciu protokołu MIDI, pages 40–52. Monografie Katedry Automatyzacji Procesów. Katedra Automatyzacji Procesów, Akademia Górniczo-Hutnicza w Krakowie, 2015.

[145] A. Czyżewski, B. Kostek, and S. Zieliński. Synthesis of organ pipe sound based on simplified physical models. *Archives of Acoustics*, 21(2):131–147, 1996.

[146] A. R. da Silva, G. P. Scavone, and M. van Walstijn. Numerical simulations of fluid-structure interactions in single-reed mouthpieces. *The Journal of the Acoustical Society of America*, 122(3):1798–1809, Sept. 2007.

[147] M. A. Dablain. The application of high-order differencing to the scalar wave equation. *Geophysics*, 51(1):54–66, Jan. 1986.

[148] J.-P. Dalmont, J. Gilbert, and S. Ollivier. Nonlinear characteristics of single-reed instruments: Quasistatic volume flow and reed opening measurements. *The Journal of the Acoustical Society of America*, 114(4):2253–2262, Oct. 2003.

[149] S. D'Angelo. *Virtual Analog Modeling of Nonlinear Musical Circuits*. PhD thesis, Aalto University, 2014.

[150] R. B. Dannenberg. Interpolation Error in Waveform Table Lookup. In *Proceedings of the International Computer Music Conference, San Francisco*, 1998.

[151] R. B. Dannenberg. Concatenative Synthesis Using Score-Aligned Transcriptions. In *Proceedings of the International Computer Music Conference*, pages 352–355, 2006.

[152] A. de Cheveigné and N. Henrich. Fundamental Frequency Estimation of Musical Sounds. *The Journal of the Acoustical Society of America*, 111:2416, 2002.

[153] A. de Cheveigné and H. Kawahara. YIN, a fundamental frequency estimator for speech and music. *The Journal of the Acoustical Society of America*, 111(4):1917–1930, 2002.

[154] G. De Poli. A Tutorial on Digital Sound Synthesis Techniques. *Computer Music Journal*, 7(4):8–26, 1983.

[155] G. De Poli. Frequency Dependent Waveshaping. In *Proceedings of the International Computer Music Conference*, pages 91–101, 1984.

[156] G. De Poli, A. Piccialli, and C. Roads. *Representations of Musical Signals*. The MIT Press, 1991.

[157] R. J. Delekta and M. Pluta. Synthesis System for Wind Instruments Parts of the Symphony Orchestra. In B. Borkowski, editor, *Proceedings of 7th Forum Acusticum*. The Polish Acoustical Society, 2014.

[158] R. J. Delekta and M. Pluta. Implementacja reguł wykonawczych w syntezie dźwięku instrumentów dętych zmodyfikowaną metodą samplingową [An implementation of the performance rules in the modified sampling synthesis of wind instruments]. In Z. Kulka and M. Lewandowski, editors, *Materiały XVI Międzynarodowego Sympozjum Inżynierii i Reżyserii Dźwięku ISSET'2015 [Proceedings of XVI International Symposium on Sound Engineering and Tonmeistering ISSET'2015]*, pages 119–125, 2015.

[159] R. J. Delekta, L. J. Spałek, and M. Pluta. The impact of selected parameters of a modified sampling synthesis on the result of its auditory assessment. *Journal of Applied Mathematics and Physics*, 4(2):221–226, 2016.

[160] A. K. Dewdney. Computer recreations: A cellular universe of debris, droplets, defects and demons. *Scientific American*, pages 88–91, Aug. 1989.

[161] A. Di Scipio. Composition by exploration of nonlinear dynamical systems. In *Proceedings of the International Computer Music Conference. San Francisco*, pages 324–327, 1990.

[162] A. Di Scipio. Interactive micro-time sound design. *Journal of Electroacoustic Music*, 1997.

[163] R. W. Dickey. Infinite systems of nonlinear oscillation equations related to the string. *Proceedings of the American Mathematical Society*, 23(3):459–459, Mar. 1969.

[164] R. W. Dickey. Stability of periodic solutions of the nonlinear string. *Quarterly of Applied Mathematics*, 38(2):253–259, July 1980.

[165] P. H. Dietz and N. Amir. Synthesis of trumpet tones by physical modeling. In *Proceedings of the International Symposium on Musical Acoustics (ISMA), Dourdan*, pages 472–477, 1995.

[166] P. Djoharian. Shape and Material Design in Physical Modeling Sound Synthesis. In *Proceedings of the International Computer Music Conference (ICMC), Berlin*, pages 38–45. ICMA, 2000.

[167] C. Dodge. *Current Directions in Computer Music Research*, chapter On Speech Songs, pages 9–17. MIT Press, 1989.

[168] M. Dolson. The Phase Vocoder: A Tutorial. *Computer Music Journal*, 10(4):14–27, 1986.

[169] A. Dorochowicz and B. Kostek. A Study on of Music Features Derived from Audio Recordings Examples – a Quantitative Analysis. *Archives of Acoustics*, 43(3):505–516, 2018.

[170] A. Douglas. *Electronic Music Production*. Pitman, 1973.

[171] J. Driedger, T. Prätzlich, and M. Müller. Let it Bee – Towards NMF-inspired Audio Mosaicing. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, pages 350–356, 2015.

[172] K. Droba. *Kompozytorzy polscy 1918–2000: praca zbiorowa. 1: Eseje*, chapter Sonoryzm polski. Akademia Muzyczna im. Stanisława Moniuszki w Gdańsku, 2005.

[173] V. Dubos, J. Kergomard, A. Khettabi, J.-P. Dalmont, D. H. Keefe, and C. J. Nederveen. Theory of Sound Propagation in a Duct with a Branched Tube Using Modal Decomposition. *Acta Acustica united with Acustica*, 85(2):153–169, 1999.

[174] E. Ducasse. A Physical Model of a Single-Reed Wind Instrument, Including Actions of the Player. *Computer Music Journal*, 27(1):59–70, Mar. 2003.

[175] H. Dudley. Remaking Speech. *The Journal of the Acoustical Society of America*, 11(2):169–177, 1939.

[176] H. Dudley. Fundamentals of Speech Synthesis. *Journal of the Audio Engineering Society*, 3(4):170–185, 1955.

[177] H. Dudley, R. Riesz, and S. Watkins. A synthetic speaker. *Journal of the Franklin Institute*, 227(6):739–764, 1939.

[178] M. Dziubiński and B. Kostek. Octave Error Immune and Instantaneous Pitch Detection Algorithm. *Journal of New Music Research*, 34(3):273–292, Nov. 2005.

[179] L. Döbereiner. Models of Constructed Sound: Nonstandard Synthesis as an Aesthetic Perspective. *Computer Music Journal*, 35(3):28–39, Sept. 2011.

[180] G. Eckel, F. Iovino, and R. Causseé. Sound synthesis by physical modelling with Modalys. In *Proceedings of the International Symposium on Musical Acoustics*, pages 479–482, 1995.

[181] D. P. W. Ellis. Beat Tracking by Dynamic Programming. *Journal of New Music Research*, 36(1):51–60, 2007.

[182] J. Engel, C. Resnick, A. Roberts, S. Dieleman, D. Eck, K. Simonyan, and M. Norouzi. Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders. *ArXiv*, 2017.

[183] C. Erkut. Aspects in analysis and model-based sound synthesis of plucked string instruments, 2002.

[184] C. Erkut, M. Karjalainen, P. Huang, and V. Välimäki. Acoustical analysis and model-based sound synthesis of the kantele. *The Journal of the Acoustical Society of America*, 112(4):1681–1691, Oct. 2002.

[185] D. Ernst. *The Evolution of Electronic Music*. Schirmer Books, 1977.

[186] G. Essl, S. Serafin, P. R. Cook, and J. O. Smith III. Musical Applications of Banded Waveguides. *Computer Music Journal*, 28(1):51–63, Mar. 2004.

[187] G. Essl, S. Serafin, P. R. Cook, and J. O. Smith III. Theory of Banded Waveguides. *Computer Music Journal*, 28(1):37–50, Mar. 2004.

[188] C. W. Farrow. A continuously variable digital delay element. In *Proceedings of IEEE International Symposium on Circuits and Systems, Espoo*, pages 2641–2645, 1988.

[189] J. L. Flanagan. *Speech Analysis Synthesis and Perception*. Springer-Verlag, 1972.

[190] J. L. Flanagan and R. M. Golden. Phase Vocoder. *Bell System Technical Journal*, 45(9):1493–1509, 1966.

[191] N. H. Fletcher and T. D. Rossing. *The Physics of Musical Instruments*. Springer-Verlag GmbH, second edition, 1998.

[192] B. Fornberg. *A Practical Guide to Pseudospectral Methods*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2009.

[193] A. Franck. Arbitrary sample rate conversion with resampling filters optimized for combination with oversampling. In *Proceedings of 2011 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 149–152, 2011.

[194] D. J. Freed. Waveshaping Analysis and Implentation: A Generalized Approach Using Complex Arithmetic. In *Proceedings of the International Computer Music Conference*, pages 193–234, 1983.

[195] A. Friberg, R. Bresin, and J. Sundberg. Overview of the KTH rule system for musical performance. *Advances in Cognitive Psychology*, 2(2–3):145–161, 2006.

[196] S. Fukayama and M. Goto. HarmonyMixer: Mixing the Character of Chords among Polyphonic Audio. In *Proceedings of the International Computer Music Conference (ICMC)*, 2014.

[197] D. Gabor. Theory of communication. *Journal of the Institution of Electrical Engineers – Part III*, 93(26):429–457, 1946.

[198] D. Gabor. Acoustical quanta and the theory of hearing. *Nature*, 159(4044):591–594, 1947.

[199] A. Gabrielsson. Interplay between analysis and synthesis in studies of music performance and music experience. *Music Perception: An Interdisciplinary Journal*, 3(1):59–86, Oct. 1985.

[200] P. Galluzzo, J. Woodhouse, and H. Mansour. Assessing Friction Laws for Simulating Bowed-String Motion. *Acta Acustica united with Acustica*, 103(6):1080–1099, Nov. 2017.

[201] B. Gaster, L. Howes, D. R. Kaeli, P. Mistry, and D. Schaa. *Heterogeneous Computing with OpenCL*. Morgan Kaufmann, 2011.

[202] S. A. Gelfand. *Hearing*. Taylor & Francis Inc, 2009.

[203] J. E. Gentle. *Random Number Generation and Monte Carlo Methods*. Springer, second edition, 2003.

[204] M. Gerhardt, H. Schuster, and J. J. Tyson. A cellular automaton model of excitable media III. Fitting the Belousov-Zhabotinskii reaction. *Physica D*, 46(3):416–426, 1990.

[205] B. Geveci and J. D. A. Walker. Nonlinear resonance of rectangular plates. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 457(2009):1215–1240, May 2001.

[206] F. S. Gillan and S. J. Elliott. Measurement of the Torsional Modes of Vibration of Strings on Instruments of the Violin Family. *Journal of Sound and Vibration*, 130(2):347–351, Apr. 1989.

[207] J. Gleick. *Chaos: Making a New Science*. Cardinal, London, 1988.

[208] T. W. Goeddel and S. C. Bass. High-quality synthesis of musical voices in discrete time. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(3):623–633, 1984.

[209] M. Gogins. Iterated functions systems music. *Computer Music Journal*, 15(1):40–48, 1991.

[210] M. Gogins. Gabor synthesis of recurrent iterated function systems. In *Proceedings of the International Computer Music Conference*, pages 349–350, 1995.

[211] A. Gołaś and R. Filipek. Digital Synthesis of Sound Generated by Tibetan Bowls and Bells. *Archives of Acoustics*, 41(1):139–150, Mar. 2016.

[212] E. Gómez, A. Klapuri, and B. Meudic. Melody description and extraction in the context of music content processing. *Journal of New Music Research*, 32(1):23–40, 2003.

[213] M. Good. Lessons from the Adoption of MusicXML as an Interchange Standard. In *Proceedings of XML, Boston*, 2006.

[214] J. W. Gordon and J. M. Strawn. *Digital Audio Signal Processing: An Anthology*, chapter An introduction to the phase vocoder, pages 221–270. William Kauffmann, Inc., 1985.

[215] C. Gough. The nonlinear free vibration of a damped elastic string. *The Journal of the Acoustical Society of America*, 75(6):1770–1776, June 1984.

[216] F. Gouyon and P. Herrera. A Beat Induction Method For Musical Audio Signals. In *Proceedings of 4th WIAMIS – Special session on Audio Segmentation and Digital Music, London*, 2003.

[217] F. Gouyon, A. Klapuri, S. Dixon, M. Alonso, G. Tzanetakis, C. Uhle, and P. Cano. An experimental comparison of audio tempo induction algorithms. *IEEE Transactions on Speech and Audio Processing*, 14(5):1832–1844, 2006.

[218] A. Gołaś, R. Olszewski, and M. Ryś. Analiza elementu zawieszenia silnika z wykorzystaniem pakietu ANSYS [An suspension engine element analysis with use of the ANSYS package]. *Logistyka*, 2009(6):1–10, 2009.

[219] K. F. Graff. *Wave Motion in Elastic Solids*. Dover Publications, revised edition, 1991.

[220] M. J. Grant. *Serial Music, Serial Aesthetics: Compositional Theory in Post-War Europe*. Cambridge University Press, 2005.

[221] A. Guadamuz-Gonzalez. Viral contracts or unenforceable documents? Contractual validity of copyleft licenses. *European Intellectual Property Review*, 26(8):331–339, 2004.

[222] P. Guillemain. A Digital Synthesis Model of Double-Reed Wind Instruments. *EURASIP Journal on Advances in Signal Processing*, 2004(7), June 2004.

[223] P. Guillemain, J. Kergomard, and T. Voinier. Real-time synthesis of clarinet-like instruments using digital impedance models. *The Journal of the Acoustical Society of America*, 118(1):483–494, July 2005.

[224] B.-Z. Guo and W. Guo. Adaptive stabilization for a Kirchhoff-type nonlinear beam under boundary output feedback control. *Nonlinear Analysis: Theory, Methods & Applications*, 66(2):427–441, Jan. 2007.

[225] K. Gustafson. Domain Decomposition, Operator Trigonometry, Robin Condition. *Contemporary Mathematics*, 218:432–437, 1998.

[226] J. K. Hahn, J. Geigel, J. W. Lee, L. Gritz, T. Takala, and S. Mishra. An Integrated Approach to Motion and Sound. *Journal of Visualization and Computer Animation*, 6(2):109–123, 1995.

[227] S. Hainsworth and M. Macleod. Onset Detection in Musical Audio Signals. In *Proceedings of the International Computer Music Conference (ICMC), Singapore*, 2003.

[228] J. Hajda. A New Model for Segmenting the Envelope of Musical Signals: The Relative Salience of Steady State Versus Attack, Revisited. *Journal of the Audio Engineering Society*, 1996.

[229] D. E. Hall. Piano string excitation II: General solution for a hard narrow hammer. *The Journal of the Acoustical Society of America*, 81(2):535–546, Feb. 1987.

[230] D. E. Hall. Piano string excitation III: General solution for a soft narrow hammer. *The Journal of the Acoustical Society of America*, 81(2):547–555, Feb. 1987.

[231] F. J. Harris. On the Use of Windows for Harmonic Analysis With the Discrete Fourier Transform. *Proceedings of the IEEE*, 66(1):51–83, 1978.

[232] F. J. Harris. Performance and design of Farrow filter used for arbitrary resampling. In *Proceedings of the 13th International Conference on Digital Signal Processing, Santorini*, pages 595–599, 1997.

[233] W. M. Hartmann. Pitch, periodicity, and auditory organization. *The Journal of the Acoustical Society of America*, 100(6):3491–3502, Dec. 1996.

[234] E. Hermanowicz, M. Rojewski, and M. Blok. A sample rate converter based on a fractional delay filter bank. In *Proceedings of ICSPAT, Dallas*, 2000.

[235] L. Hiller and P. Ruiz. Synthesizing Musical Sounds by Solving the Wave Equation for Vibrating Objects: Part 1. *Journal of the Audio Engineering Society*, 19(6):462–470, June 1971.

[236] L. Hiller and P. Ruiz. Synthesizing Musical Sounds by Solving the Wave Equation for Vibrating Objects: Part 2. *Journal of the Audio Engineering Society*, 19(7):542–551, July 1971.

[237] A. Hirschberg, J. Gilbert, R. Msallam, and A. P. J. Wijnands. Shock waves in trombones. *The Journal of the Acoustical Society of America*, 99(3):1754–1758, Mar. 1996.

[238] S. Hirschman. *Digital Waveguide Modeling and Simulation of Reed Woodwind Instruments*. PhD thesis, Stanford University, 1991.

[239] M. D. Hoffman, P. R. Cook, and D. M. Blei. Bayesian Spectral Matching: Turning Young MC into MC Hammer via MCMC Sampling. In *Proceedings of the International Computer Music Conference (ICMC), Montreal*, 2009.

[240] P. Hoffmann. Towards an 'automated art': Algorithmic processes in Xenakis' compositions. *Contemporary Music Review*, 21(2–3):121–131, 2002.

[241] A. V. Holden. *Chaos*. Princeton University Press, 1986.

[242] F. Holm. Understanding FM Implementations: A Call for Common Standards. *Computer Music Journal*, 16(1):34–42, 1992.

[243] S. R. Holtzman. A Description of an Automatic Digital Sound Synthesis Instrument. Technical Report 59, Edinburgh: Department of Artificial Intelligence, 1978.

[244] S. R. Holtzman. An Automated Digital Sound Synthesis Instrument. *Computer Music Journal*, 3(2):53–61, 1979.

[245] M. A. Horn. Nonlinear boundary stabilization of a von Kármán plate via bending moments only. In *System Modelling and Optimization*, pages 706–715. Springer, Berlin–Heidelberg, 1994.

[246] A. Horner. Wavetable Matching Synthesis of Dynamic Instruments with Genetic Algorithms. *Journal of the Audio Engineering Society*, 43(11):916–931, 1995.

[247] A. Horner. Double-Modulator FM Matching of Instrument Tones. *Computer Music Journal*, 20(2):57–71, 1996.

[248] A. Horner. A Comparison of Wavetable and Fm Parameter Spaces. *Computer Music Journal*, 21(4):55–85, 1997.

[249] A. Horner, J. Beauchamp, and L. Haken. Wavetable and FM matching synthesis of musical instrument tones. In *Proceedings of the 1992 International Computer Music Conference, San Francisco*, pages 18–21, 1992.

[250] A. Horner, J. Beauchamp, and L. Haken. FM Matching Synthesis with Genetic Algorithms. *Computer Music Journal*, 17(4):17–29, 1993.

[251] A. Horner, J. Beauchamp, and L. Haken. Methods for Multiple Wavetable Synthesis of Musical Instrument Tones. *Journal of the Audio Engineering Society*, 41(5):336–356, 1993.

[252] H. S. Howe. *Electronic Music Synthesis*. Norton, 1975.

[253] S. Howell. The Lost Art Of Sampling. *Sound on Sound*, 2005. Online, http://www.soundonsound.com/techniques/lost-art-sampling-part-1, accessed 2017.07.21.

[254] B. Hsu and M. Sosnick-Pérez. Real-time GPU audio. *Communications of the ACM*, 56(6):54–62, June 2013.

[255] N. Hu. *Automatic Construction of Synthetic Musical Instruments and Performers.* PhD thesis, Carnegie Mellon University, 2013.

[256] D. M. Huber and R. E. Runstein. *Modern Recording Techniques.* Taylor & Francis Ltd., 2013.

[257] T. E. Hull and A. R. Dobell. Random Number Generators. *SIAM Review*, 4(3):230–254, 1962.

[258] B. A. Hutchins. The Frequency Modulation Spectrum of an Exponential Voltage-Controlled Oscillator. *Journal of the Audio Engineering Society*, 23(3):200–206, 1975.

[259] B. A. Hutchins Jr. Experimental Electronic Music Devices Employing Walsh Functions. *Journal of the Audio Engineering Society*, 21(8):640–645, 1973.

[260] R. Ierusalimschy, L. H. de Figueiredo, and W. C. Filho. Lua – An Extensible Extension Language. *Software: Practice and Experience*, 26(6):635–652, 1996.

[261] S. Ilanko. Vibration and Post-buckling of In-Plane Loaded Rectangular Plates Using a Multiterm Galerkin's Method. *Journal of Applied Mechanics*, 69(5):589–592, 2002.

[262] M. Ishibashi. Electronic musical instrument, 1987.

[263] P. Iverson and C. L. Krumhansl. Isolating the dynamic attributes of musical timbre. *The Journal of the Acoustical Society of America*, 94(5):2595–2603, 1993.

[264] D. A. Jaffe and J. O. Smith III. Extensions of the Karplus-Strong Plucked-String Algorithm. *Computer Music Journal*, 7(2):56–68, 1983.

[265] S. G. James. Developing a flexible and expressive realtime polyphonic wave terrain synthesis instrument based on a visual and multidimensional methodology. Master's thesis, Edith Cowan University, 2005.

[266] S. G. James. Spectromorphology and Spatiomorphology of Sound Shapes: Audio-Rate AEP and DBAP Panning of Spectra. In *Proceedings of the International Computer Music Conference, Texas*, pages 278–285, 2015.

[267] S. G. James. *Spectromorphology and Spatiomorphology: Wave Terrain Synthesis as a framework for controlling Timbre Spatialisation in the Frequency Domain.* PhD thesis, Edith Cowan University, 2015.

[268] J. Janer. *Singing-driven Interfaces for Sound Synthesizers.* PhD thesis, Universitat Pompeu Fabra, 2008.

[269] J. Janer and M. de Boer. Extending voice-driven synthesis to audio mosaicing. In *Proceedings of the 5th Sound and Music Computing Conference, Berlin*, 2008.

[270] T. Jehan. Event-Synchronous Music Analysis/Synthesis. In *Proceedings of the COST-G6 Conference on Digital Audio Effects (DAFx), Naples*, pages 361–366, 2004.

[271] C. R. Johnson, W. A. Sethares Jr, and A. G. Klein. *Software Receiver Design: Build Your Own Digital Communication System in Five Easy Steps*. Cambridge University Press, 2011.

[272] J. M. Johnson and A. K. Bajaj. Amplitude modulated and chaotic dynamics in resonant motion of strings. *Journal of Sound and Vibration*, 128(1):87–107, Jan. 1989.

[273] D. L. Jones and T. W. Parks. Generation and Combination of Grains for Music Synthesis. *Computer Music Journal*, 12(2):27–34, 1988.

[274] K. Jones. Compositional Applications of Stochastic Processes. *Computer Music Journal*, 5(2):45–61, 1981.

[275] J. Justice. Analytic signal processing in music computation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 27(6):670–684, 1979.

[276] W. Kaegi. A minimum description of the linguistic sign repertoire (first part). *Interface*, 2(2):141–156, 1973.

[277] W. Kaegi. A minimum description of the linguistic sign repertoire (part two). *Interface*, 3(2):137–158, 1974.

[278] W. Kaegi and S. Tempelaars. VOSIM – a new sound synthesis system. *Journal of the Audio Engineering Society*, 26(6):418–426, 1978.

[279] H. Kaprykowsky and X. Rodet. Globally Optimal Short-Time Dynamic Time Warping, Application to Score to Audio Alignment. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, 2006.

[280] H. Kaprykowsky and X. Rodet. Musical Alignment Using Globally Optimal Short-Time Dynamic Time Warping. In *Proceedings of DAGA: Fortschritte der Akustik, Stuttgart*, pages 837–838, 2007.

[281] M. Karjalainen. Block-compiler: Efficient simulation of acoustic and audio systems. In *114th Audio Engineering Society Convention, Amsterdam*, 2003.

[282] M. Karjalainen. Time-domain physical modeling and real-time synthesis using mixed modeling paradigms. In *Proceedings of the Stockholm Musical Acoustics Conference, Volume 1*, pages 393–396, 2003.

[283] M. Karjalainen and C. Erkut. Digital Waveguides versus Finite Difference Structures: Equivalence and Mixed Modeling. *EURASIP Journal on Advances in Signal Processing*, 2004(7):978–989, June 2004.

[284] M. Karjalainen and U. K. Laine. A model for real-time sound synthesis of guitar on a floating-point signal processor. In *Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 3653–3656, 1991.

[285] M. Karjalainen, V. Välimäki, and Z. Jánosy. Towards High-Quality Sound Synthesis of the Guitar and String Instruments. In *Proceedings of International Computer Music Conference (ICMC), Tokyo*, pages 56–63, 1993.

[286] K. Karplus and A. Strong. Digital Synthesis of Plucked-String and Drum Timbres. *Computer Music Journal*, 7(2):43–55, 1983.

[287] D. H. Keefe. Experiments on the single woodwind tone hole. *The Journal of the Acoustical Society of America*, 72(3):688–699, Sept. 1982.

[288] D. H. Keefe. Theory of the single woodwind tone hole. *The Journal of the Acoustical Society of America*, 72(3):676–687, Sept. 1982.

[289] D. H. Keefe. Woodwind air column models. *The Journal of the Acoustical Society of America*, 88(1):35–51, July 1990.

[290] D. H. Keefe. Physical Modeling of Wind Instruments. *Computer Music Journal*, 16(4):57–73, 1992.

[291] D. Keller and C. Rolfe. The Corner Effect. In *Proceedings of the XII Colloquium on Musical Informatics*, 1998.

[292] D. Keller and B. Truax. Ecologically-based granular synthesis. In *Proceedings of the International Computer Music Conference, San Francisco*, pages 117–120, 1998.

[293] H. Kenmochi and H. Ohshita. VOCALOID – Commercial singing synthesizer based on sample concatenation. In *Proceeding of the Interspeech*, 2007.

[294] R. G. Keys. Cubic convolution interpolation for digital image processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(6):1153–1160, Dec. 1981.

[295] G. Kirchhoff. *Vorlesungen über mathematische Physik. Mechanik.* Teubner, Leipzig, 1876.

[296] A. Kirke and E. R. Miranda. *Guide to Computing for Expressive Music Performance.* Springer London, 2013.

[297] D. H. Klatt. Review of text-to-speech conversion for English. *The Journal of the Acoustical Society of America*, 82(3):737–793, Sept. 1987.

[298] P. Kleczkowski. Group Additive Synthesis. *Computer Music Journal*, 13(1):12–20, Sept. 1990.

[299] P. Kleczkowski. *Percepcja dźwięku [Perception of sound].* Wydawnictwa AGH, 2013.

[300] M. K. Klingbeil. *Spectral Analysis, Editing, and Resynthesis: Methods and Applications*. PhD thesis, Columbia University, New York, NY, USA, 2009.

[301] D. E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, third edition, 1997.

[302] R. Kobayashi. Sound Clustering Synthesis Using Spectral Data. In *Proceedings of the International Computer Music Conference (ICMC), Singapore*, 2003.

[303] R. Kobayashi. Real-time Force-based Sound Synthesis Using GPU Parallel Computing. In *Proceedings of the 19th International Conference on Digital Audio Effects (DAFx-16), Brno*, pages 153–157, 2016.

[304] G. M. Koenig. Project 1: a programme for musical composition. *Electronic Music Reports*, 2:32–44, 1970.

[305] G. M. Koenig. Project 2: a programme for musical composition. *Electronic Music Reports*, 3:1–16, 1970.

[306] K. Konopko. Zastosowanie procesorów masowo-równoległych w addytywnej syntezie sygnałów. *Przegląd Elektrotechniczny*, 90(11):231–234, 2014.

[307] K. Kowalczyk and M. von Walstijn. Modeling Frequency-Dependent Boundaries as Digital Impedance Filters in FDTD and K-DWM Room Acoustics Simulations. *Journal of the Audio Engineering Society*, 56(7/8):569–583, 2008.

[308] T. Kreger. Real-time Cellular Automata Filters Implemented with Max/MSP. In *Proceedings of the Australasian Computer Music Conference*, 1999.

[309] U. R. Kristiansen and E. M. Viggen. *Computational Methods in Acoustics*. Department of Electronics and Telecommunications – Norwegian University of Science and Technology, 2010.

[310] R. Kronland-Martinet and A. Grossmann. *Representations of musical signals*, chapter Application of time-frequency and time-scale methods (wavelet transforms) to the analysis, synthesis, and transformation of natural sounds, pages 45–85. MIT Press, 1991.

[311] E. V. Kurmyshev. Transverse and longitudinal mode coupling in a free vibrating soft string. *Physics Letters A*, 310(2–3):148–160, Apr. 2003.

[312] T. I. Laakso, V. Välimäki, M. Karjalainen, U. K. Laine, and T. Tolonen. Splitting the Unit Delay: Tools for Fractional Delay Filter Design. *IEEE Signal Processing Magazine*, 13(1):30–60, 1996.

[313] J. A. Laird. *The physical modelling of drums using digital waveguides*. PhD thesis, University of York, 2001.

[314] C. Lambourg, A. Chaigne, and D. Matignon. Time-domain simulation of damped impacted plates. II. Numerical model and results. *The Journal of the Acoustical Society of America*, 109(4):1433–1447, Apr. 2001.

[315] M. Lang and T. I. Laakso. Simple and robust method for the design of allpass filters using least-squares phase error criterion. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 41(1):40–48, 1994.

[316] C. G. Langton. Computation at the Edge of Chaos: Phase Transitions and Emergent Computation. *Physica D*, 42(1–3):12–37, 1990.

[317] P. Lansky. Linear prediction: The hard, but interesting way to do things. In *Audio Engineering Society Conference: 5th International Conference: Music and Digital Technology*, May 1987.

[318] P. Lansky. *Current Directions in Computer Music Research*, chapter Compositional Applications of Linear Predictive Coding, pages 5–8. MIT Press, Cambridge, MA, USA, 1989.

[319] P. Lansky and K. Steiglitz. Synthesis of Timbral Families by Warped Linear Prediction. *Computer Music Journal*, 5(3):45–49, 1981.

[320] J. Laroche. Frequency-Domain Techniques For High-Quality Voice Modification. In *Proceedings of the DAFx (Digital Audio Effects) Conference, London*, pages 328–332, 2003.

[321] O. Lartillot and P. Toiviainen. A Matlab Toolbox for Musical Feature Extraction from Audio. In *Proceeding of the 10th International Conference on Digital Audio Effects (DAFx), Bordeaux*, 2007.

[322] M. Laurson, C. Erkut, V. Välimäki, and M. Kuuskankare. Methods for Modeling Realistic Playing in Acoustic Guitar Synthesis. *Computer Music Journal*, 25(3):38–49, Sept. 2001.

[323] M. Laurson, V. Valimaki, and C. Erkut. Production of Virtual Acoustic Guitar Music. In *Proceedings of the Audio Engineering Society Conference: 22nd International Conference: Virtual, Synthetic, and Entertainment Audio*, pages 249–255, June 2002.

[324] A. Lazier and P. Cook. Mosievius: Feature Driven Interactive Audio Mosaicing. In *Proceeding of the 6th International Conference on Digital Audio Effects (DAFx-03), London, UK*, 2003.

[325] V. Lazzarini, J. Timoney, and T. Lysaght. Asymmetric-Spectra Methods for Adaptive Fm Synthesis. In *Proceedings of the 11th International Conference on Digital Audio Effects (DAFx), Espoo*, 2008.

[326] V. Lazzarini, J. Timoney, and T. Lysaght. The Generation of Natural-Synthetic Spectra by Means of Adaptive Frequency Modulation. *Computer Music Journal*, 32(2):9–22, 2008.

[327] M. LeBrun. A Derivation of the Spectrum of FM with a Complex Modulating Wave. *Computer Music Journal*, 1(4):51–52, 1977.

[328] M. LeBrun. Digital Waveshaping Synthesis. *Journal of the Audio Engineering Society*, 27(4):250–266, 1979.

[329] P. L'Ecuyer. Tables of Linear Congruential Generators of Different Sizes and Good Lattice Structure. *Mathematics of Computation*, 68:249–260, 1999.

[330] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

[331] K. A. Legge and N. H. Fletcher. Nonlinear generation of missing modes on a vibrating string. *The Journal of the Acoustical Society of America*, 76(1):5–12, July 1984.

[332] A. W. Leissa. *Vibration of Shells*. Acoustical Society of America, 1993.

[333] J. Leonard and C. Cadoz. Physical Modelling Concepts for a Collection of Multisensory Virtual Musical Instruments. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME'15), Baton Rouge, USA*, pages 150–155, May 2015.

[334] R. J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2002.

[335] S. M. Lim and B. T. G. Tan. Performance of the Genetic Annealing Algorithm in DFM Synthesis of Dynamicmusical Sound Samples. *Journal of the Audio Engineering Society*, 47(5):339–354, 1999.

[336] E. Lindemann. Music Synthesis with Reconstructive Phrase Modeling. *IEEE Signal Processing Magazine*, 24(2):80–91, Mar. 2007.

[337] A. T. Lindsay, A. P. Parkes, and R. A. Fitzgerald. Description-driven context-sensitive effects. In *Proceedings of the COST-G6 Conference on Digital Audio Effects (DAFx), London*, pages 350–353, 2003.

[338] S. R. Livingstone, R. Muhlberger, A. R. Brown, and W. F. Thompson. Changing Musical Emotion: A Computational Rule System for Modifying Score and Performance. *Computer Music Journal*, 34(1):41–64, Mar. 2010.

[339] J. London. *Hearing in Time: Psychological Aspects of Musical Meter*. Oxford University Press, Sept. 2004.

[340] D. Lorrain. A Panoply of Stochastic 'Cannons'. *Computer Music Journal*, 4(1):53–81, 1980.

[341] M. W. Macon, L. Jensen-Link, J. Oliverio, M. A. Clements, and E. B. George. Concatenation-based MIDI-to-Singing Voice Synthesis. In *Proceedings of the 103rd Meeting of the AES*, 1997.

[342] E. Maestre and E. Gómez. Automatic Characterization of Dynamics and Articulation of Monophonic Expressive Recordings. In *Procedings of the 118th AES Convention*, 2005.

[343] E. Maestre, A. Hazan, R. Ramirez, and A. Perez. Using Concatenative Synthesis for Expressive Performance in Jazz Saxophone. In *Proceedings of the International Computer Music Conference, San Francisco*, pages 219–222, 2006.

[344] E. Maestre, R. Ramírez, S. Kersten, and X. Serra. Expressive concatenative synthesis by reusing samples from real performance recordings. *Computer Music Journal*, 33(4):23–42, Dec. 2009.

[345] P. C. Mahalanobis. On the generalised distance in statistics. *Proceedings of the National Institute of Sciences of India*, 2(1):49–55, 1936.

[346] L. Majkut. Quantitative analysis of phase trajectory as the information about technical condition of the object. *Acta Physica Polonica. A*, 121(1-A):A–168–A–173, 2012.

[347] L. Majkut and R. Olszewski. Acoustic Eigenanalysis with Radial Basis Functions. *Acta Physica Polonica A*, 125(4A):A–77–A–83, apr 2014.

[348] L. Majkut and R. Olszewski. Detekcja uszkodzeń przekładni obiegowych z wykorzystaniem modelu dynamicznego zazębienia [Planetary gears system faults detection using dynamic model of meshing gear]. *Logistyka*, 2014(3):4099–4105, 2014.

[349] L. Majkut, R. Olszewski, and M. Pluta. Comparison of three meshless methods in plane acoustic eigenanalysis. In *Proceedings of the 7th Forum Acusticum, Kraków*, 2014.

[350] J. Makhoul. Linear prediction: A tutorial review. *Proceedings of the IEEE*, 63(4):561–580, Apr. 1975.

[351] S. G. Mallat and Z. Zhang. Matching Pursuits with Time-Frequency Dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, 1993.

[352] H. Mansour, J. Woodhouse, and G. P. Scavone. Enhanced Wave-Based Modelling of Musical Strings. Part 2: Bowed Strings. *Acta Acustica united with Acustica*, 102(6):1094–1107, Nov. 2016.

[353] H. Mansour, J. Woodhouse, and G. P. Scavone. On Minimum Bow Force for Bowed Strings. *Acta Acustica united with Acustica*, 103(2):317–330, 2017.

[354] G. Marino, M.-H. Serra, and J.-M. Raczinski. The UPIC System: Origins and Innovations. *Perspectives of New Music*, 31(1):258–269, 1993.

[355] J. E. Markel and A. H. Gray. *Linear Prediction of Speech*. Springer-Verlag, New York, 1982.

[356] J. Martínez and J. A. Agulló. Conical bores. Part I: Reflection functions associated with discontinuities. *The Journal of the Acoustical Society of America*, 84(5):1613–1619, Nov. 1988.

[357] M. Mathews and L. Rosler. Graphical Language for the Scores of Computer-Generated Sounds. *Perspectives of New Music*, 6(2):92–118, 1968.

[358] R. J. McAulay and T. F. Quatieri. Speech analysis/synthesis based on a sinusoidal representation. *IEEE Transactions on Acoustics Speech and Signal Processing*, 34(4):744–754, 1986.

[359] M. McIntyre and J. Woodhouse. On the Fundamentals of Bowed-String Dynamics. *Acustica*, 43:93–108, 1979.

[360] M. E. McIntyre, R. T. Schumacher, and J. Woodhouse. On the oscillations of musical instruments. *The Journal of the Acoustical Society of America*, 74(5):1325–1345, 1983.

[361] N. McLaren. Synthetic Sound on Film. *Journal of the Society of Motion Picture Engineers*, 50(3):233–247, 1948.

[362] Y. Medan, E. Yair, and D. Chazan. Super Resolution Pitch Determination of Speech Signals. *IEEE Transactions on Signal Processing*, 39(1):40–48, 1991.

[363] Y. Meron. *High Quality Singing Synthesis Using the Selection-based Synthesis Scheme*. PhD thesis, University of Tokyo, 1999.

[364] Merriam-Webster. Online Dictionary. `https://www.merriam-webster.com/`, 2018. Accessed: 2018-10-18.

[365] J. Michalczyk and P. Czubak. Influence of collisions with a material feed on cophasal mutual synchronisation of driving vibrators of vibratory machines. *Journal of Theoretical and Applied Mechanics*, 48(1):155–172, 2010.

[366] H. Mikelson. Terrain Mapping with Dynamic Surfaces. *The Csound Magazine*, 2000.

[367] H. Mikelson. *The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing, and Programming*, chapter Terrain Mapping Synthesis. MIT Press, 2000.

[368] D. Millen. Cellular Automata Music. In *Proceedings of the International Computer Music Conference (ICMC), Glasgow*, pages 314–316, 1990.

[369] A. Mills and R. C. de Souza. Gestural Sounds by Means of Wave Terrain Synthesis. In *Proceedings of the VI Brazilian Symposium on Computer Music*, volume 3, pages 9–16, 1999.

[370] E. R. Miranda. Cellular Automata Music Investigation. Master's thesis, University of York, 1990.

[371] E. R. Miranda. Cellular Automata Synthesis of Acoustic Particles. *Proceedings of the International Computer Music Conference*, 1995:233–234, 1995.

[372] E. R. Miranda. Evolving Cellular Automata Music: From Sound Synthesis to Composition. In *Proceedings of 6th European Conference on Artificial Life, Prague*, 2001.

[373] E. R. Miranda. *Computer Sound Design: Synthesis techniques and programming.* Focal Press, second edition, 2002.

[374] E. R. Miranda and J. A. Biles, editors. *Evolutionary Computer Music.* Springer, 2007.

[375] E. R. Miranda, A. Kirke, and Q. Zhang. Artificial Evolution of Expressive Performance of Music: An Imitative Multi-Agent Systems Approach. *Computer Music Journal*, 34(1):80–96, Mar. 2010.

[376] S. Mishra and J. K. Hahn. Mapping Motion to Sound and Music in Computer Animation and VE. In *Proceedings of the Pacific Graphics, Seoul*, pages 83–98, 1995.

[377] T. J. Mitchell. *An Exploration of Evolutionary Computation Applied to Frequency Modulation Audio Synthesis Parameter Optimisation.* PhD thesis, University of the West of England, Bristol, 2010.

[378] Y. Mitsuhashi. Audio Synthesis by Functions of Two Variables. *Journal of the Audio Engineering Society*, 30(10):701–706, 1982.

[379] Y. Mitsuhashi. Piecewise Interpolation Technique for Audio Signal Synthesis. *Journal of the Audio Engineering Society*, 30(4):192–202, 1982.

[380] F. Mittelbach, M. Goossens, J. Braams, D. Carlisle, and C. Rowley. *The LaTeX Companion (Tools and Techniques for Computer Typesetting).* Addison-Wesley Professional, 2004.

[381] D. Moelants. Statistical Analysis of Written and Performed Music. A Study of Compositional Principles and Problems of Coordination and Expression in "Punctual" Serial Music. *Journal of New Music Research*, 29(1):37–60, Mar. 2000.

[382] A. Moles. *Information Theory and Esthetic Perception.* Urbana: University of Illinois Press, 1968.

[383] R. A. Moog. A Voltage-Controlled Low-Pass High-Pass Filter for Audio Signal Processing. In *Audio Engineering Society Convention 17*, Oct. 1965.

[384] F. C. Moon. *Chaotic Vibrations: An Introduction for Applied Scientists and Engineers.* New York Wiley-Interscience, 1987.

[385] B. Moore. *An Introduction to the Psychology of Hearing.* Brill Academic Pub, 2013.

[386] F. R. Moore. Table lookup noise for sinusoidal digital oscillators. *Computer Music Journal*, 1(2):26–29, 1977.

[387] J. A. Moorer. The use of the phase vocoder in computer music applications. *Journal of the Audio Engineering Society*, 26(1/2):42–45, 1978.

[388] J. A. Moorer. The Use of Linear Prediction of Speech in Computer Music Applications. *Journal of the Audio Engineering Society*, 27(3):134–140, 1979.

[389] D. Morrill. Trumpet Algorithms For Computer Composition. *Computer Music Journal*, 1(1):46–52, 1977.

[390] J. D. Morrison and J.-M. Adrien. MOSAIC: A Framework for Modal Synthesis. *Computer Music Journal*, 17(1):45–56, 1993.

[391] P. M. Morse and K. U. Ingard. *Theoretical Acoustics*. Princeton University Press, 1987.

[392] E. Motuk, R. Woods, and S. Bilbao. FPGA-based hardware for physical modelling sound synthesis by finite difference schemes. In *Proceedings of IEEE International Conference on Field-Programmable Technology, Singapore*. IEEE, 2005.

[393] E. Motuk, R. Woods, S. Bilbao, and J. McAllister. Design Methodology for Real-Time FPGA-Based Sound Synthesis. *IEEE Transactions on Signal Processing*, 55(12):5833–5845, Dec. 2007.

[394] R. Msallam, S. Dequidt, R. Caussé, and S. Tassart. Physical Model of the Trombone Including Nonlinear Effects. Application to the Sound Synthesis of Loud Tones. *Acta Acustica united with Acustica*, 86(4):725–736, 2000.

[395] J. Mullen, D. M. Howard, and D. T. Murphy. Waveguide physical modeling of vocal tract acoustics: flexible formant bandwidth control from increased model dimensionality. *IEEE Transactions on Audio, Speech and Language Processing*, 14(3):964–971, May 2006.

[396] A. Munshi, B. Gaster, T. G. Mattson, J. Fung, and D. Ginsburg. *OpenCL Programming Guide*. Addison-Wesley Professional, 2011.

[397] A. D. Muradova. Numerical techniques for linear and nonlinear eigenvalue problems in the theory of elasticity. In *Proceedings of the 12th Biennial Computational Techniques and Applications Conference, Melbourne*, 2004.

[398] M. Müller. *Information Retrieval for Music and Motion*. Springer, 2007.

[399] J. Nam, V. Välimäki, J. S. Abel, and J. O. Smith III. Efficient antialiasing oscillator algorithms using low-order fractional delay filters. *IEEE Transactions on Audio, Speech and Language Processing*, 18(4):773–785, 2010.

[400] R. Narasimha. Non-Linear vibration of an elastic string. *Journal of Sound and Vibration*, 8(1):134–146, July 1968.

[401] E. Narmour. *The Analysis and Cognition of Basic Melodic Structures: The Implication-Realization Model*. University of Chicago Press, 1990.

[402] E. Narmour. *The Analysis and Cognition of Melodic Complexity: The Implication-Realization Model*. University of Chicago Press, 1992.

[403] A. H. Nayfeh and D. T. Mook. *Nonlinear Oscillations*. Wiley-VCH Verlag GmbH, second revised edition, May 1995.

[404] C. J. Nederveen. *Acoustical Aspects of Woodwind Instruments*. Northern Illinois University Press, revised edition, 1998.

[405] C. J. Nederveen, J. Jansen, and R. van Hassel. Corrections for Woodwind Tone-Hole Calculations. *Acta Acustica united with Acustica*, 84(5):957–966, Sept. 1998.

[406] H.-W. Nienhuys and J. Nieuwenhuizen. LilyPond, a system for automated music engraving. In *Proceedings of the XIV Colloquium on Musical Informatics (CIM), Firenze*, 2003.

[407] J. O. D. Noreland. A Numerical Method for Acoustic Waves in Horns. *Acta Acustica united with Acustica*, 88(4):576–586, July 2002.

[408] B. J. Noye and M. J. Rankovic. An accurate explicit finite diference technique for solving the one-dimensional wave equation. *Communications in Applied Numerical Methods*, 2(6):557–561, Nov. 1986.

[409] B. J. Noye and M. J. Rankovic. An accurate five-point implicit finite difference method for solving the one-dimensional wave equation. *Communications in Applied Numerical Methods*, 5(4):247–252, May 1989.

[410] C. Ó Nuanáin, P. Herrera, and S. Jordà. An Evaluation Framework and Case Study for Rhythmic Concatenative Synthesis. In *Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR)*, pages 67–72, 2016.

[411] C. Ó Nuanáin, S. Jordà, and P. Herrera. k-Best Hidden Markov Model Decoding for Unit Selection in Concatenative Sound Synthesis. In *Proceedings of the International Symposium on Computer Music Multidisciplinary Research*, 2017.

[412] S. Ochiai, Y. Yamaguchi, and Y. Kodama. The Flexible Sound Synthesizer on an FPGA. In *First International Symposium on Computing and Networking*. IEEE, Dec. 2013.

[413] P. A. V. Olivares. *Acoustic wave propagation and modeling turbulent water flow with acoustics for district heating pipes*. PhD thesis, Uppsala Universitet, 2009.

[414] J. P. Olive. Rule synthesis of speech from dyadic units. In *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing*, pages 568–570, 1977.

[415] D. Oram. *An Individual Note of Music, Sound and Electronics*. Anomie Publishing, second edition, 2016.

[416] N. Orio and D. Schwarz. Alignment of Monophonic and Polyphonic Music to a Score. In *Proceedings of the International Computer Music Conference (ICMC), Havana*, 2001.

[417] R. Orton, A. Hunt, and R. Kirk. Graphical Control of Granular Synthesis Using a Cellular Automata and the Freehand Program. In *Proceedings of the 1991 International Computer Music Conference, San Francisco*, pages 416–418, 1991.

[418] D. Overholt. The MATRIX: a novel controller for musical expression. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 38–41, 2001.

[419] D. Overholt. New Musical Mappings for the MATRIX Interface. In *Proceedings of the International Computer Music Conference, Gothenburg*, 2002.

[420] A. Ozga. The Effect of Pulse Amplitudes on Quality of Determining Distribution of Pulses Forcing Vibration of an Damped Oscillator. *Acta Physica Polonica A*, 128(1A):A–67–A–71, July 2015.

[421] O. O'Reilly and P. J. Holmes. Non-linear, non-planar and non-periodic vibrations of a string. *Journal of Sound and Vibration*, 153(3):413–435, Mar. 1992.

[422] L. O'Sullivan. Development of a Graphical Sound Synthesis Controller Exploring Cross-Modal Perceptual Analogies. Master's thesis, Trinity College Dublin, 2007.

[423] J. Pakarinen, V. Välimäki, and M. Karjalainen. Physics-Based Methods for Modeling Nonlinear Vibrating Strings. *Acta Acustica united with Acustica*, 91(2):312–325, Mar. 2005.

[424] J.-P. Palamin, P. Palamin, and A. Ronveaux. A Method of Generating and Controlling Musical Asymmetrical Spectra. *Journal of Audio Engineering Society*, 36(9):671–685, 1988.

[425] J. Pan, X. Li, J. Tian, and T. Lin. Short sound decay of ancient Chinese music bells. *The Journal of the Acoustical Society of America*, 112(6):3042–3045, Dec. 2002.

[426] G. Peeters. A large set of audio features for sound description (similarity and classification) in the Cuidado project. Technical report, Ircam – Centre Pompidou, Paris, 2004.

[427] H. Penttinen, C. Erkut, J. Pölkki, V. Välimäki, and M. Karjalainen. Design and analysis of a modified kantele with increased loudness. *Acta Acustica united with Acustica*, 91(2):261–268, 2005.

[428] H. Penttinen, J. Pakarinen, V. Välimäki, M. Laurson, H. Li, and M. Leman. Model-based sound synthesis of the guqin. *The Journal of the Acoustical Society of America*, 120(6):4052–4063, Dec. 2006.

[429] G. K. Percival. *Physical Modelling meets Machine Learning: Performing Music with a Virtual String Ensemble*. PhD thesis, University of Glasgow, May 2013.

[430] G. E. Peterson and H. L. Barney. Control Methods Used in a Study of the Vowels. *The Journal of the Acoustical Society of America*, 24:175–184, 1952.

[431] G. E. Peterson, W. S. Wang, and E. Sivertsen. Segmentation Techniques in Speech Synthesis. *The Journal of the Acoustical Society of America*, 30:739–742, 1958.

[432] S. Petrausch, J. Escolano, and R. Rabenstein. A General Approach to Block-Based Physical Modeling with Mixed Modeling Strategies for Digital Sound Synthesis. In *Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages 21–24, 2005.

[433] S. Petrausch and R. Rabenstein. Tension modulated nonlinear 2D models for digital sound synthesis with the functional transformation method. In *Proceedings of EUSIPCO-05, Thirteenth European Signal Processing Conference, Antalya*, Sept. 2005.

[434] F. Pfeifle and R. Bader. Performance Controller for Physical Modelling FPGA Sound Synthesis of Musical Instruments. In *Sound – Perception – Performance*, pages 331–350. Springer International Publishing, 2013.

[435] D. K. Phillips and A. Purvis. Multirate Additive Synthesis. *Computer Music Journal*, 23(1):28–40, Mar. 1999.

[436] A. Piccialli, S. Cavaliere, I. Ortosecco, and P. Basile. Modifications of natural sounds using a pitch synchronous technique. In *Proceedings of the International Workshop on Models and Representations of Musical Signals*, 1992.

[437] R. Pitteroff and J. Woodhouse. Mechanics of the Contact Area Between a Violin Bow and a String. Part II: Simulating the Bowed String. *Acta Acustica united with Acustica*, 84(4):744–757, 1998.

[438] R. Plomp. *The Intelligent Ear: On the Nature of Sound Perception*. Psychology Press, 2013.

[439] M. Pluta. *Badanie interakcji słuchacza ze sprzętowo-programowym systemem symulowania błędów intonacyjnych [Research on interactions between a listener and a hardware-software system for simulating pitch intonation errors]*. PhD thesis, AGH University of Science and Technology, Kraków, 2008.

[440] M. Pluta. Physical Modeling Synthesis on Heterogeneous Platforms. In B. Borkowski, editor, *Proceedings of 7th Forum Acusticum*. The Polish Acoustical Society, 2014.

[441] M. Pluta. *Aktualności inżynierii akustycznej i biomedycznej*, chapter Korpus automatycznego syntezatora frazy dla instrumentów dętych orkiestry symfonicznej, pages 41–51. Polskie Towarzystwo Akustyczne Oddział w Krakowie, 2018.

[442] M. Pluta, B. Borkowski, I. Czajka, and K. Suder-Dębska. Sound Synthesis Using Physical Modeling on Heterogeneous Computing Platforms. *Acta Physica Polonica A*, 128(1–A):22–28, July 2015.

[443] M. Pluta and R. J. Delekta. *Progress of Acoustics 2015*, chapter Technique to Seamlessly Connect Sound Samples in Sampling Synthesis, pages 271–282. Polish Acoustical Society Wroclaw Division, 2015.

[444] M. Pluta and P. Kleczkowski. A software system for off-site timbre solfege with remote results management capability. In A. Józefczak, editor, *Proceedings of 59th Open Seminar on Acoustics*, pages 209–212, 2012.

[445] M. Pluta, L. J. Spałek, and R. J. Delekta. An Automatic Synthesis of Musical Phrases from Multi-Pitch Samples. *Archives of Acoustics*, 42(2):235–247, Jan. 2017.

[446] M. Pluta, L. J. Spałek, and R. J. Delekta. A modified sampling synthesis for a realistic simulation of wind instruments – the design and implementation. *Journal of Applied Mathematics and Physics*, 4(2):215–220, 2016.

[447] P. Polotti. A Pitch-Synchronous Extension of Fractal Additive Synthesis via Time-Varying Cosine Modulated Filter Banks. *IEEE Signal Processing Letters*, 15:433–436, 2008.

[448] P. Polotti and G. Evangelista. Fractal Additive Synthesis via Harmonic-Band Wavelets. *Computer Music Journal*, 25(3):22–37, 2001.

[449] M. R. Portnoff. Implementation of the digital phase vocoder using the fast Fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 24(3):243–248, June 1976.

[450] Y. Potard, P.-F. Baisnèe, and J.-B. Barrière. Experimenting with models of resonance produced by a new technique for the analysis of impulsive sounds. In P. Berg, editor, *Proceedings of the 1986 International Computer Music Conference, San Francisco*, pages 269–274, 1986.

[451] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in FORTRAN. The Art of Scientific Computing*. Cambridge University Press, second edition, 1993.

[452] R. Prudon. *A Selection/Concatenation TTS Synthesis System*. PhD thesis, LIMSI, University of Paris XI, 2003.

[453] R. Rabenstein, S. Petrausch, A. Sarti, G. De Sanctis, C. Erkut, and M. Karjalainen. Block-Based Physical Modeling for Digital Sound Synthesis. *IEEE Signal Processing Magazine*, 24(2):42–54, Mar. 2007.

[454] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, 1989.

[455] L. R. Rabiner and B.-H. Juang. *Fundamentals of Speech Recognition*. Prentice Hall, 1993.

[456] L. R. Rabiner and R. W. Schafer. *Digital Processing of Speech Signals*. Pearson, 1978.

[457] R. Ramirez, A. Hazan, and E. Maestre. A Tool for Generating and Explaining Expressive Music Performances of Monophonic Jazz Melodies. *International Journal on Artificial Intelligence Tools*, 15(4):673–691, 2006.

[458] R. Ramirez, E. Maestre, A. Pertusa, E. Gomez, and X. Serra. Performance-Based Interpreter Identification in Saxophone Audio Recordings. *IEEE Transactions on Circuits and Systems for Video Technology*, pages 356–364, 2007.

[459] E. Rank and G. Kubin. A waveguide model for slapbass synthesis. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 443–446, 1997.

[460] J. S. Rao. *Dynamics of Plates*. CRC Press, 1998.

[461] L. Rhaouti, A. Chaigne, and P. Joly. Time-domain modeling and numerical simulation of a kettledrum. *The Journal of the Acoustical Society of America*, 105(6):3545–3562, June 1999.

[462] T. Rhea. *The evolution of electronic musical instruments in the United States*. PhD thesis, Nashville: George Peabody College for Teachers, 1972.

[463] J. S. Risberg. Non-Linear Estimation of FM Synthesis Parameters. In *Proceedings of the 67th Convention of the Audio Engineering Society. New York*, 1980.

[464] J.-C. Risset. Computer music experiments, 1964–... *Computer Music Journal*, 9(1):11–18, 1985.

[465] J.-C. Risset. Pitch and rhythm paradoxes: Comments on "Auditory paradox based on fractal waveform". *The Journal of the Acoustical Society of America*, 80(3):961–962, Sept. 1986.

[466] J. E. M. Rivera, H. P. Oquendo, and M. L. Santos. Asymptotic behavior to a von Kármán plate with boundary memory conditions. *Nonlinear Analysis: Theory, Methods & Applications*, 62(7):1183–1205, Sept. 2005.

[467] C. Roads. An interview with Gottfried Michael Koenig. *Computer Music Journal*, 2(3):11–15, 1978.

[468] C. Roads. Automated granular synthesis of sound. *Computer Music Journal*, 2(2):61–62, 1978.

[469] C. Roads. *Representations of Musical Signals*, chapter Asynchronous granular synthesis, pages 143–185. The MIT Press, 1991.

[470] C. Roads. *The Computer Music Tutorial*. The MIT Press, 1996.

[471] C. Roads. Sound composition with pulsars. *Journal of the Audio Engineering Society*, 49(3):134–147, 2001.

[472] C. Roads. *Microsound*. The MIT Press, 2004.

[473] A. Röbel. Transient detection and preservation in the phase vocoder. In *Proceedings of the International Computer Music Conference (ICMC), Singapore*, 2003.

[474] X. Rodet and P. Depalle. Spectral Envelopes and Inverse FFT Synthesis. In *Audio Engineering Society Convention 93*, Oct. 1992.

[475] X. Rodet, P. Depalle, and G. Poirot. Diphone Sound Synthesis Based on Spectral Envelopes and Harmonic/Noise Excitation Functions. In *Proceedings of the International Computer Music Conference, San Francisco*, pages 313–321. International Computer Music Association, 1988.

[476] X. Rodet and F. Jaillet. Detection and modeling of fast attack transients. In *Proceedings of the International Computer Music Conference (ICMC), Havana*, 2001.

[477] X. Rodet, Y. Potard, and J.-B. Barrière. The CHANT Project: From the Synthesis of the Singing Voice to Synthesis in General. *The Computer Music Journal*, 8(3):15–31, 1984.

[478] X. Rodet and C. Vergez. Nonlinear Dynamics in Physical Models: From Basic Models to True Musical-Instrument Models. *Computer Music Journal*, 23(3):35–49, Sept. 1999.

[479] T. D. Rossing and N. H. Fletcher. Nonlinear vibrations in plates and gongs. *The Journal of the Acoustical Society of America*, 73(1):345–351, Jan. 1983.

[480] S. E. Roucos and A. M. Wilgus. High quality time-scale modification for speech. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 493–496, 1985.

[481] J. B. Rovan, M. M. Wanderley, S. Dubnov, and P. Depalle. Instrumental Gestural Mapping Strategies as Expressivity Determinants in Computer Music Performance. In *Proceedings of the Workshop "Kansei – The Technology of Emotion"*, 1997.

[482] D. R. Rowland. Parametric resonance and nonlinear string vibrations. *American Journal of Physics*, 72(6):758–766, June 2004.

[483] M. B. Rubin and O. Gottlieb. Numerical solutions of forced vibration and whirling of a non-linear string using the theory of a Cosserat point. *Journal of Sound and Vibration*, 197(1):85–101, Oct. 1996.

[484] P. M. Ruiz. A technique for simulating the vibrations of strings with a digital computer. Master's thesis, University of Ilinois, 1969.

[485] M. Russ. *Sound Synthesis and Sampling.* Taylor & Francis Ltd, third edition, 2008.

[486] J. Rychlik and M. Pluta. An impact of selected simulation qualities on the intonation aspect of pitch perception in the sound of orchestra instruments. In Z. Damijan, J. Wiciak, and C. Kasprzak, editors, *Proceedings of XX Conference on Acoustic and Biomedical Engineering*, pages 170–171, Apr. 2013.

[487] M. Rymar. Realizacja syntezatora dźwiękowego w formie wtyczki VST [Implementation of a sound synthesizer as a VST plug-in]. Engineer's thesis, Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie, 2017.

[488] D. Salomon. *Assemblers and Loaders.* Ellis Horwood, 1993.

[489] F. Salzer and C. Schachter. *Counterpoint in Composition: The Study of Voice Leading.* Columbia University Press, 1989.

[490] L. Savioja, V. Välimäki, and J. O. Smith III. Real-time additive synthesis with one million sinusoids using a GPU. In *the AES 128th International Convention*, London, UK, 2010.

[491] G. Scavone. *An Acoustic Analysis of Single-Reed Woodwind Instruments with an Emphasis on Design and Performance Issues and Digital Waveguide Techniques.* PhD thesis, Department of Music, Stanford University, 1997.

[492] R. A. Schaefer. Electronic Musical Tone Production by Nonlinear Waveshaping. *Journal of the Audio Engineering Society*, 18(4):413–417, 1970.

[493] D. Schaeffer and M. Golubitsky. Boundary Conditions and Mode Jumping in the Buckling of a Rectangular Plate. *Communications in Mathematical Physics*, 69(3):209–236, Oct. 1979.

[494] R. W. Schafer and L. R. Rabiner. A digital signal processing approach to interpolation. In *Proceedings of the IEEE*, volume 61, pages 692–702, 1973.

[495] W. Schottstaedt. The simulation of natural instrument tones using frequency modulation with a complex modulation wave. *Computer Music Journal*, 1(4):46–50, 1977.

[496] M. Schroeder. Vocoders: analysis and synthesis of speech. *Proceedings of the IEEE*, 54(5):720–734, 1966.

[497] M. Schroeder and B. Atal. Generalized Short-Time Power Spectra and Autocorrelation Functions. *Journal of the Acoustical Society of America*, 34(11):1679–1683, 1962.

[498] E. Schubert, J. Wolfe, and A. Tarnopolsky. Spectral Centroid and Timbre in Complex, Multiple Instrumental Textures. In *Proceedings of the 8th International Conference on Music Perception & Cognition (ICMPC)*, pages 654–657, 2004.

[499] R. Schwartz, J. Klovstad, J. Makhoul, D. Klatt, and V. Zue. Diphone synthesis for phonetic vocoding. In *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing*, pages 891–894, 1979.

[500] D. Schwarz. A system for data-driven concatenative sound synthesis. In *Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-00), Verona*, 2000.

[501] D. Schwarz. The Caterpillar System for Data-Driven Concatenative Sound Synthesis. In *Proceeding of the Digital Audio Effects (DAFx), London, UK*, pages 135–140, 2003.

[502] D. Schwarz. *Data-Driven Concatenative Sound Synthesis*. PhD thesis, University of Paris 6 – Pierre et Marie Curie, 2004.

[503] D. Schwarz. Current research in Concatenative Sound Synthesis. In *Proceedings of the International Computer Music Conference (ICMC)*, 2005.

[504] D. Schwarz, G. Beller, B. Verbrugghe, and S. Britton. Real-Time Corpus-Based Concatenative Synthesis with CataRT. In *Proceedings of the 9th International Conference on Digital Audio Effects (DAFx-06)*, pages 279–282, 2006.

[505] D. Schwarz, R. Cahen, and S. Britton. Principles and Applications of Interactive Corpus-Based Concatenative Synthesis. In *Proceedings of the Journées d'Informatique Musicale (JIM)*, Mar. 2008.

[506] S. Serafin, F. Avanzini, and D. Rocchesso. Bowed String Simulation Using an Elasto-plastic Friction Model. In *Proceedings of the Stockholm Music Acoustics Conference (SMAC 03)*, volume 1, pages 95–98, Aug. 2003.

[507] S. Serafin and J. O. Smith III. A Multirate, Finite-width, Bow-String Interaction Model. In *Proceedings of the COST-G6 Digital Audio Effects Conference, Verona*, pages 207–210, 2000.

[508] S. Serafin, J. O. Smith III, and J. Woodhouse. An Investigation of the Impact of Torsion Waves and Friction Characteristics on the Playability of Virtual Bowed Strings. In *Proceedings of the 1999 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics. WASPAA'99, New York*, pages 87–99, 1999.

[509] M.-H. Serra. Stochastic Composition and Stochastic Timbre: GENDY3 by Iannis Xenakis. *Perspectives of New Music*, 31(1):236–257, 1992.

[510] M.-H. Serra, D. Rubine, and R. Dannenberg. Analysis and Synthesis of Tones by Spectral Interpolation. *Journal of the Audio Engineering Society*, 38(3):111–128, 1990.

[511] X. Serra. *A System for Sound Analysis/Transformation/Synthesis based on a Deterministic plus Stochastic Decomposition*. PhD thesis, Department of Music, Stanford University, 1989.

[512] X. Serra and J. O. Smith III. Spectral Modeling Synthesis: A Sound Analysis/Synthesis System Based on a Deterministic Plus Stochastic Decomposition. *Computer Music Journal*, 14(4):12–24, 1990.

[513] I. Simon, S. Basu, D. Salesin, and M. Agrawala. Audio Analogies: Creating New Music from an Existing Performance by Concatenative Synthesis. In *Proceedings of the International Computer Music Conference*, pages 65–72, 2005.

[514] D. Slater. Chaotic Sound Synthesis. *Computer Music Journal*, 22(2):12–19, 1998.

[515] D. Smalley. *The Language of Electroacoustic Music*, chapter Spectromorphology and Structuring Processes, pages 61–93. Macmillan Press Ltd., 1986.

[516] S. Smallwood, D. Trueman, P. R. Cook, and G. Wang. Composing for Laptop Orchestra. *Computer Music Journal*, 32(1):9–25, 2008.

[517] P. Smaragdis, C. Févotte, G. J. Mysore, N. Mohammadiha, and M. Hoffman. Static and Dynamic Source Separation Using Nonnegative Factorizations: A Unified View. *IEEE Signal Processing Magazine*, 31(3):66–75, 2014.

[518] P. Smaragdis, B. Raj, and M. Shashanka. Sparse and shift-invariant feature extraction from non-negative data. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing ICASSP*, pages 2069–2072, 2008.

[519] J. O. Smith III. Efficient simulation of the reed-bore and bow-string mechanisms. In *Proceedings of the International Computer Music Conference, The Hague*, pages 275–280, 1986.

[520] J. O. Smith III. Music applications of digital waveguides. Technical report, Department of Music, Stanford University, 1987.

[521] J. O. Smith III. Viewpoints on the History of Digital Synthesis. In *Proceedings of the International Computer Music Conference ICMC-91*, pages 1–10, 1991.

[522] J. O. Smith III. Physical Modeling Using Digital Waveguides. *Computer Music Journal*, 16(4):74–91, 1992.

[523] J. O. Smith III. Efficient synthesis of stringed musical instruments. In *Proceedings of the International Computer Music Conference, Tokyo*, pages 64–71, 1993.

[524] J. O. Smith III. *Musical Signal Processing*, chapter Acoustic modeling using digital waveguides, pages 221–264. Swets & Zeitlinger, 1997.

[525] J. O. Smith III. *Introduction to Digital Filters with Audio Applications*. W3K Publishing, http://www.w3k.org/books/, 2007.

[526] J. O. Smith III. *Physical Audio Signal Processing: for Virtual Musical Instruments and Digital Audio Effects*. W3K Publishing, 2010.

[527] J. O. Smith III and G. P. Scavone. The one-filter Keefe clarinet tonehole. In *Proceedings of 1997 Workshop on Applications of Signal Processing to Audio and Acoustics*. IEEE, 1997.

[528] J. O. Smith III and S. Van Duyne. Commuted piano synthesis. In *Proceedings of the International Computer Music Conference, Banff*, pages 335–342, 1995.

[529] T. Sorensen, A. F. Donaldson, M. Batty, G. Gopalakrishnan, and Z. Rakamarić. Portable inter-workgroup barrier synchronisation for GPUs. *ACM SIGPLAN Notices*, 51(10):39–58, Oct. 2016.

[530] M. Sosnick and W. Hsu. Efficient finite difference-based sound synthesis using GPUs. In *Proceedings of the Sound and Music Computing Conference*, 2010.

[531] M. Sosnick and W. Hsu. Implementing a Finite Difference-Based Real-time Sound Synthesizer using GPUs. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 264–267, 2011.

[532] F. Soulez, X. Rodet, and D. Schwarz. Improving Polyphonic and Poly-Instrumental Music to Score Alignment. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR), Baltimore*, 2003.

[533] J. Spałek. *Wstęp do fizyki materii skondensowanej*. Wydawnictwo Naukowe PWN, 2015.

[534] M. Spytek. Implementacja syntezatora dźwięku z zastosowaniem niekonwencjonalnych technik sterowania [Implementation of the sound synthesizer with the application of unconventional control techniques]. Master's thesis, AGH University of Science and Technology, 2016.

[535] K. Steiglitz. *A Digital Signal Processing Primer*. Addison-Wesley, 1996.

[536] T. Stilson and J. O. Smith III. Alias-Free Digital Synthesis of Classic Analog Waveforms. In *Proceedings of International Computer Music Conference, Hong Kong*, pages 332–335, 1996.

[537] J. M. Strawn. *Modeling musical transitions*. PhD thesis, Department of Music, Stanford University, June 1985.

[538] J. C. Strikwerda. *Finite Difference Schemes and Partial Differential Equations*. Society for Industrial & Applied Mathematics (SIAM), second edition, Jan. 2004.

[539] A. Stulov. Hysteretic model of the grand piano hammer felt. *The Journal of the Acoustical Society of America*, 97(4):2577–2585, Apr. 1995.

[540] B. L. Sturm, C. Roads, A. McLeran, and J. J. Shynk. Analysis, Visualization, and Transformation of Audio Signals Using Dictionary-based Methods. *Journal of New Music Research*, 38(4):325–341, 2009.

[541] B. L. T. Sturm. *Sparse Approximation and Atomic Decomposition: Considering Atom Interactions in Evaluating and Building Signal Representations*. PhD thesis, University of California, 2009.

[542] C. R. Sullivan. Extending the Karplus-Strong Algorithm to Synthesize Electric Guitar Timbres with Distortion and Feedback. *Computer Music Journal*, 14(3):26–37, 1990.

[543] A. Świercz and J. Żera. Model of Auditory Filters and MPEG-7 Descriptors in Sound Recognition. In *Proceeding of the 10th International Conference: Active Media Technology (AMT)*, pages 200–211, 2014.

[544] R. Szilard. *Theories and Applications of Plate Analysis: Classical, Numerical and Engineering Methods*. John Wiley & Sons Inc., 2004.

[545] N. Szilas and C. Cadoz. Analysis Techniques for Physical Modeling Networks. *Computer Music Journal*, 22(3):33–48, 1998.

[546] B. T. G. Tan and S. M. Lim. Automated Parameter Optimization for Double mboxFrequency-Modulation Synthesis Using the Genetic Annealing Algorithm. *Journal of the Audio Engineering Society*, 44(1–2):3–15, 1996.

[547] A. Tarczynski, W. Kozinski, and G. D. Cain. Sampling rate conversion using fractional-sample delay. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP-94*, pages 285–288, 1994.

[548] S. Tempelaars. The VOSIM oscillator. In *Proceedings of the International Computer Music Conference, Cambridge, Massachusetts*, 1976.

[549] L. H. Thomas. Elliptic problems in linear difference equations over a network. Technical report, Watson Scientific Computing Laboratory, Columbia University, New York, 1949.

[550] O. Thomas, C. Touzé, and A. Chaigne. Non-linear vibrations of free-edge thin spherical shells: modal interaction rules and 1:1:2 internal resonance. *International Journal of Solids and Structures*, 42(11-12):3339–3373, June 2005.

[551] J. Timoney and V. Lazzarini. Exponential Frequency Modulation Bandwidth Criterion for Virtual Analog Applications. In *Proceedings of the 14th International Conference on Digital Audio Effects (DAFx-11), Paris*, 2011.

[552] J. Timoney, V. Lazzarini, and T. Lysaght. A Modified FM Synthesis Approach to Bandlimited Signal Generation. In *Proceedings of the 11th International Conference on Digital Audio Effects (DAFx), Espoo*, 2008.

[553] J. Timoney, V. Lazzarini, T. Lysaght, J. Kleimola, and L. MacManus. Considerations for a Computational Estimation of the Complex FM Spectrum. In *Proceedings of the Irish Sound Science and Technology Conference, Limerick*, 2011.

[554] J. Timoney, V. Lazzarini, J. Pekonen, and V. Välimäki. Spectrally rich phase distortion sound synthesis using an allpass filter. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Taipei*, pages 293–296, 2009.

[555] D. Tokarczyk. A guitar playing robot. Master's thesis, AGH University of Science and Technology, 2014.

[556] D. Tokarczyk, M. Pluta, and J. Wiciak. Guitar Playing Robot, a Tool Assisting Instrumental Research. *Mechanics and Control*, 33(3):75–82, 2014.

[557] T. Tolonen, V. Välimäki, and M. Karjalainen. *Evaluation of Modern Sound Synthesis Methods*. Helsinki University of Technology, 1998.

[558] T. Tolonen, V. Välimäki, and M. Karjalainen. Modeling of Tension Modulation Nonlinearity in Plucked Strings. *IEEE Transactions on Speech and Audio Processing*, 8(3):300–310, May 2000.

[559] N. Tomisawa. Tone production method for an electronic musical instrument, 1981.

[560] L. N. Trefethen. *Spectral Methods in MATLAB*. Software, Environments, Tools. SIAM: Society for Industrial and Applied Mathematics, 2001.

[561] B. Truax. Organizational Techniques for C:M Ratios in Frequency Modulation. *Computer Music Journal*, 1(4):39–45, 1977.

[562] B. Truax. Real-time granulation of sampled sound with the DMX-1000. In *Proceedings of the 1987 International Computer Music Conference, San Francisco*, pages 138–145, 1987.

[563] B. Truax. Real-time granular synthesis with a digital signal processing computer. *Computer Music Journal*, 12(2):14–26, 1988.

[564] C. Truchet, G. Assayag, and P. Codognet. Visual and Adaptive Constraint Programming in Music. In *Proceedings of the International Computer Music Conference, Havana*, 2001.

[565] J. Tuomela. A note on high order schemes for the one dimensional wave equation. *BIT Numerical Mathematics*, 35(3):394–405, Sept. 1995.

[566] S. Vaidhyanathan, A. Minai, and M. Helmuth. ca: A System for Granular Processing of Sound Using Cellular Automata. In *Proceedings of the 2nd COST G-6 Workshop on Digital Audio (DAFx), Trondheim*, 1999.

[567] H. Valbret, E. Moulines, and J.-P. Tubach. Voice transformation using PSOLA technique. *Speech Communication*, 11(2–3):189–194, 1992.

[568] C. Valette. *Mechanics of Musical Instruments*, chapter The mechanics of vibrating strings, pages 116–183. Springer, New York, 1995.

[569] V. Välimäki. *Discrete-time modeling of acoustic tubes using fractional delay filters*. PhD thesis, Helsinki University of Technology, 1995.

[570] V. Välimäki, S. Bilbao, J. Pakarinen, J. O. Smith III, J. Abel, and D. Berners. *DAFX: Digital Audio Effects*, chapter Virtual Analog Effects, pages 473–550. John Wiley & Sons, Inc., second edition, 2011.

[571] V. Välimäki and C. Erkut. Commuted Waveguide Synthesis of the Clavichord. *Computer Music Journal*, 27(1):71–82, Mar. 2003.

[572] V. Välimäki, R. Hänninen, and M. Karjalainen. An Improved Digital Waveguide Model of Flute – Implementation Issues. In *Proceedings of the International Computer Music Conference, Hong Kong*, pages 1–4, 1996.

[573] V. Välimäki and A. Huovilainen. Oscillator and Filter Algorithms for Virtual Analog Synthesis. *Computer Music Journal*, 30(2):19–31, 2006.

[574] V. Välimäki, M. Karjalainen, Z. Jànosy, and U. K. Laine. A real-time DSP implementation of a flute model. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP-92*, pages 249–252, 1992.

[575] V. Välimäki, M. Karjalainen, and T. I. Laakso. Modeling of Woodwind Bores with Finger Holes. In *Proceedings of the International Computer Music Conference, Tokyo*, pages 32–39, 1993.

[576] V. Välimäki, T. I. Laakso, M. Karjalainen, and U. K. Laine. A new computational model for the clarinet. In *Proceedings of the International Computer Music Conference*, 1992.

[577] V. Välimäki, J. Pakarinen, C. Erkut, and M. Karjalainen. Discrete-time modelling of musical instruments. *Reports on Progress in Physics*, 69(1):1–78, Oct. 2005.

[578] V. Välimäki, H. Penttinen, J. Knif, M. Laurson, and C. Erkut. Sound Synthesis of the Harpsichord Using a Computationally Efficient Physical Model. *EURASIP Journal on Advances in Signal Processing*, 69(7):934–948, June 2004.

[579] V. Välimäki, T. Tolonen, and M. Karjalainen. Plucked-String Synthesis Algorithms with Tension Modulation Nonlinearity. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing ICASSP*, pages 977–980, 1999.

[580] D. Van Brink. Waveshaping with DC Offset: A Computationally Inexpensive Method of Musical Synthesis. Master's thesis, University of California Santa Cruz, 1993.

[581] K. van den Doel and U. M. Ascher. Real-Time Numerical Solution of Webster's Equation on A Nonuniform Grid. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(6):1163–1172, Aug. 2008.

[582] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. *ArXiv*, 2016.

[583] S. Van Duyne and J. O. Smith III. A Simplified Approach to Modeling Dispersion Caused by Stiffness in Strings and Plates. In *Proceedings of the International Computer Music Conference, Aarhus*, pages 407–410, 1994.

[584] S. A. Van Duyne, J. R. Pierce, and J. O. Smith III. Traveling Wave Implementation of a Lossless Mode-coupling Filter and the Wave Digital Hammer. In *Proceedings of the International Computer Music Conference, Aarhus*, pages 411–418, 1994.

[585] S. A. Van Duyne and J. O. Smith III. Developments for the commuted piano. In *Proceedings of the 1995 International Computer Music Conference, Banff*, pages 335–343, 1995.

[586] M. van Walstijn and F. Avanzini. Modelling the mechanical response of the reed-mouthpiece-lip system of a clarinet. Part II: A lumped model approximation. *Acta Acustica united with Acustica*, 93(3):435–446, 2007.

[587] M. van Walstijn and G. P. Scavone. The Wave Digital Tonehole Model. In *Proceedings of the International Computer Music Conference, Berlin*, pages 465–468, 2000.

[588] G. Velikic, E. L. Titlebaum, and M. F. Bocko. Musical note segmentation employing combined time and frequency analyses. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 4, pages 277–280, 2004.

[589] M.-P. Verge. *Aeroacoustics of confined jets with applications to the physical modeling of recorder-like instruments*. PhD thesis, Eindhoven University of Technology, 1995.

[590] M.-P. Verge, A. Hirschberg, and R. Caussé. Sound production in recorderlike instruments. II. A simulation model. *The Journal of the Acoustical Society of America*, 101(5):2925–2939, May 1997.

[591] C. Vergez, A. Almeida, R. Caussé, and X. Rodet. Toward a simple physical model of double-reed musical instruments: influence of aero-dynamical losses in the embouchure on the coupling. *Acta Acustica united with Acustica*, 89(6):964–973, 2003.

[592] C. Vergez and X. Rodet. A new algorithm for nonlinear propagation of sound wave: Application to a physical model of a trumpet. *Journal of Signal Processing*, 4:79–88, 2000.

[593] C. Vergez and X. Rodet. Dynamic systems and physical models of trumpet-like instruments: A study and asymptotical properties. *Acta Acustica united with Acustica*, 86(1):147–162, 2000.

[594] C. Vergez and P. Tisserand. The BRASS Project, from Physical Models to Virtual Musical Instruments: Playability Issues. In *Computer Music Modeling and Retrieval*, pages 24–33. Springer, Berlin–Heidelberg, 2006.

[595] T. S. Verma, S. N. Levine, and T. H. Y. Meng. Transient Modeling Synthesis: a flexible analysis/synthesis tool for transient signals. In *Proceedings of the International Computer Music Conference*, pages 164–167, 1997.

[596] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967.

[597] Y. Wang and T. Nicol. Statistical Properties of Pseudo Random Sequences and Experiments with PHP and Debian OpenSSL. *Computers and Security*, 53:44–64, 2015.

[598] R. Waschka II and A. Kurepa. Using Fractals in Timbre Construction: An Exploratory Study. In T. Wells and D. Butler, editors, *Proceedings of the International Computer Music Conference, San Francisco*, pages 332–335, 1989.

[599] A. Watzky. Non-linear three-dimensional large-amplitude damped free vibration of a stiff elastic stretched string. *Journal of Sound and Vibration*, 153(1):125–142, Feb. 1992.

[600] H. Wen, Z. Teng, Y. Wang, and Y. Yang. Optimized Trapezoid Convolution Windows for Harmonic Analysis. *IEEE Transactions on Instrumentation and Measurement*, 62(9):2609–2612, 2013.

[601] D. Wessel, R. Felciano, A. Freed, and J. Wawrzynek. The Center for New Music and Audio Technologies. In *Proceedings of the International Computer Music Conference, San Francisco*, pages 336–339, 1989.

[602] J. Wiciak and R. Filipek. Finite element modeling of sound fields from a beam with shunted piezoceramics. In *Proceedings of the 6th European Conference on Noise Control (EURONOISE)*, 2006.

[603] G. Widmer. Modeling rational basis for musical expression. *Computer Music Journal*, 19(2):76–96, 1995.

[604] G. Widmer. Machine Discoveries: A Few Simple, Robust Local Expression Principles. *Journal of New Music Research*, 31(1):37–50, Mar. 2002.

[605] G. Widmer and A. Tobudic. Playing Mozart by Analogy: Learning Multi-level Timing and Dynamics Strategies. *Journal of New Music Research*, 32(3):259–268, Sept. 2003.

[606] N. Wiener. *The Concepts of Space and Time*, chapter Spatial-temporal continuity, quantum theory, and music, page 544. Boston: D. Reidel, 1975.

[607] G. Winham and K. Steiglitz. Input Generators for Digital Sound Synthesis. *The Journal of the Acoustical Society of America*, 47(2):665–666, 1970.

[608] E. Wójtowicz. *Sounds, Societies, Significations: Numanistic Approaches to Music*, chapter A Phenomenon of String Quartet in the Works of Kraków Composers from 1960s, pages 33–46. Cham, Switzerland: Springer International Publishing AG, 2017.

[609] S. Wolfram. Universality and Complexity in Cellular Automata. *Physica D*, 10(1–2):1–35, 1984.

[610] J. Woodhouse. Physical Modelling of Bowed Strings. *Computer Music Journal*, 16(4):43–56, 1992.

[611] I. Xenakis. *Formalized Music: Thought and Mathematics in Composition*. New York: Pendragon Press, 1992.

[612] S. Xiao and W. Feng. Inter-block GPU communication via fast barrier synchronization. In *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 1–12, Apr. 2010.

[613] A. Yardim, G. D. Cain, and P. Henry. Optimal two-term offset windowing for fractional delay. *Electronics Letters*, 32(6):526–527, 1996.

[614] A. Yardim, G. D. Cain, and A. Lavergne. Performance of fractional-delay filters using optimal offset windows. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 3, pages 2233–2236, 1997.

[615] V. Zavalishin. *The Art of VA Filter Design*. Published online, 1.1.1 edition, 2015.

[616] J. Żera. A Filter-Bank Model for the Detection of Asynchrony Between the Components of a Multitone Complex. *Acta Acustica United with Acustica*, 90(5):901–917, 2004.

[617] J. Żera. *Acoustics and Audio Technology: a Tutorial for Multimedia*. Instytut Radioelektroniki i Technik Multimedialnych WEiTI PW, 2016.

[618] Q. Zhang, L. Ye, and Z. Pan. Physically-Based Sound Synthesis on GPUs. In *Entertainment Computing – ICEC 2005*, volume 3711/2005, pages 328–333. Springer, Berlin–Heidelberg, 2005.

[619] A. Zils and F. Pachet. Musical mosaicing. In *Proceedings of the COST G-6 Conference on Digital Audio Effects (DaFx-01)*, pages 39–44, 2001.

Index