TEKNILLINEN KORKEAKOULU
Teknillisen fysiikan ja matematiikan osasto
Teknillisen fysiikan koulutusohjelma

Jarmo Hiipakka

Implementation and Control of a Real-Time Guitar Synthesizer

Diplomi-insinöörin tutkintoa varten tarkastettavaksi jätetty diplomityö

Työn valvoja	professori	Matti Karjalainen
Työn ohjaaja	dosentti	Vesa Välimäki

Espoo 19.10.1999

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ
Teknillisen fysiikan ja matematiikan osasto

Tekijä:	Jarmo Hiipakka
Osasto:	Teknillisen fysiikan ja matematiikan osasto
Pääaine:	Informaatiotekniikka
Sivuaine:	Signaalinkäsittelytekniikka ja akustiikka
Työn nimi:	Reaaliaikaisen kitarasyntetisaattorin toteutus ja ohjaus
Title in English:	Implementation and Control of a Real-Time Guitar Synthesizer
Professuuri:	Akustiikka ja äänenkäsittelytekniikka, S-89
Työn valvoja:	professori Matti Karjalainen
Työn ohjaaja:	dosentti Vesa Välimäki

Tässä diplomityössä kuvataan reaaliaikaisen kitarasyntetisaattorin toteutus ja ohjaus. Työssä toteutettu syntetisaattori perustuu akustisen kitaran fysikaaliseen malliin. Reaaliaikatoteutus on tehty yleiskäyttöisessä työasemaympäristössä olio-ohjelmoinnin lähestymistapaa käyttäen. Syntetisaattorin ohjausta varten on kehitetty uusi ohjausprotokolla.

Diplomityö esittelee akustisen kitaran perusrakenteen ja äänentuottomekanismin. Kitaran soittotekniikoita kuvataan lyhyesti painottaen erityisesti klassisen kitaran soittotekniikoita. Työssä kuvataan laskennallisesti tehokkaan, soittimen fysiikkaan perustuvan mallin kehittely. Mallinnuksessa olennainen kitaran kielen malli perustuu digitaalisten aaltojohtojen teoriaan ja on yhden viivesilmukan kielimalli. Kommutoitu aaltojohtosynteesi on menetelmä, jolla laskennalliseen malliin voidaan liittää kitaran kopan sekä kielen näppäyksen vaikutus. Lisäksi esitetään kaikukopan tärkeimpien resonanssien mallinnus erillisiä digitaalisuotimia käyttäen.

Työssä kuvattu kitaramallin reaaliaikatoteutus on tehty C++-ohjelmointikielellä. Toteutuksen osat on jaettu kahteen luokkakirjastoon. Toinen kirjasto sisältää kaikki yleiskäyttöiset signaalinkäsittelyrakenteet ja toinen sisältää erityisesti soitinmallinnusta varten tehdyt luokat. Toteutettu kitaramalli koostuu kuudesta kahteen suuntaan värähtelevästä kielimallista, jotka toimivat 22050 Hz:n näytetaajuudella, sekä kahdesta kitaran kopan resonansseja mallintavasta suotimesta, joiden näytetaajuus on 2205 Hz. Myös työasemaympäristössä tehtävän signaalinkäsittelyn yleisiä kysymyksiä on työssä käsitelty.

Synteesimallin ohjausta käsittelevän osuuden aluksi kerrataan mallin parametrit ja niiden vaikutukset mallin tuottamaan ääneen. Lisäksi kuvataan parametrien arvojen muuttamisesta aiheutuvia ongelmia, sekä esitetään ratkaisuja näihin ongelmiin. Työssä kuvataan joitakin käytössä olevia äänisynteesin ohjausprotokollia ja esitellään uusi tekstipohjainen ohjausprotokolla. Diplomityössä selostetaan myös tämän uuden hierarkkisen ohjausprotokollan soveltaminen toteutetun kitaramallin ohjaukseen.

Diplomityön lopussa esitetään yhteenveto työn pääkohdista ja esitetään vaihtoehtoja tulevan tutkimustyön suunniksi.

Sivumäärä: 74	Avainsanat: kitara, mallipohjainen äänisynteesi, synteesin ohjaus
Täytetään osastolla	
Hyväksytty:	Kirjasto:

HELSINKI UNIVERSITY OF TECHNOLOGY ABSTRACT OF MASTER'S THESIS
Department of Engineering Physics and Mathematics

Author:	Jarmo Hiipakka
Department:	Department of Engineering Physics and Mathematics
Major subject:	Information technology
Minor subject:	Signal processing and acoustics
Title:	Implementation and Control of a Real-Time Guitar Synthesizer
Title in Finnish:	Reaaliaikaisen kitarasyntetisaattorin toteutus ja ohjaus
Chair:	Acoustics and Audio Signal Processing, code S-89
Supervisor:	professor Matti Karjalainen
Instructor:	docent Vesa Välimäki

The implementation and control of a real-time guitar synthesizer are discussed in this thesis. The implemented guitar sound synthesizer is a physics-based model of the acoustic guitar. The real-time model is implemented on a workstation computer platform using object-oriented programming paradigm, and a new protocol is developed for the control of the synthesizer.

The thesis introduces the sound production principles and the structure of an acoustic guitar. The playing techniques are shortly described placing main emphasis on the classical guitar playing. A computationally efficient and physically relevant model of the acoustic guitar is then described. The model is based on the digital waveguide synthesis method, formulated as a single delay loop model. Commuted waveguide synthesis method is described as a way of including the guitar body and the plucked excitation in the model. Also digital filter implementation of the most prominent body resonances is discussed.

The guitar model implementation using C++ programming language is reported. The implementation is contained in two class libraries, one for the general purpose signal processing units and another for the special instrument model classes. The implemented synthesis model contains six dual-polarization string models running at the sampling rate of 22050 Hz, and two shared body resonators running at the rate of 2205 Hz. General issues on signal processing on a general purpose workstation platform are also discussed.

The discussion on the guitar model control starts with a review of the parameters of the synthesis model. The implications of dynamically altering the values of the model's parameters are described along with the means of reducing the severity of such artifacts. Current sound synthesis control protocols are described, a new text-based protocol with hierarchical addressing mechanism is developed, and its application to the guitar model is discussed.

The thesis is concluded with a summary of the work, and a proposal for future research directions is given.

Number of pages: 74 Keywords: guitar, model-based sound synthesis, synthesis control

Department fills

Approved:	Library code:

Preface

There may be, for ought we know, infinite inventions of art, the possibility whereof we should hardly ever believe, if they were fore-reported to us. Had we lived in some rude and remote part of the world, and should have been told, that it is possible, only by a hollow piece of wood, and the guts of beasts stirred by the fingers of men, to make so sweet and melodious a noise, we should have thought it utterly incredible: vet now, that we see and hear it ordinarily done, we make it no wonder.

— Joseph Hall (1574–1656)

This master's thesis on guitar sound synthesis has been submitted for official examination in Espoo, Finland on October 19, 1999. The work has been carried out at Helsinki University of Technology, Telecommunications Software and Multimedia Laboratory, and at the Laboratory of Acoustics and Audio Signal Processing, and it has been supported by the Academy of Finland. The basis for the work has been the previous work done by the personnel at Laboratory of Acoustics and Audio Signal Processing at HUT, during this decade.

I wish to thank Dr. Vesa Välimäki for instructing this work. His valuable comments, guidance, and encouragement have made this work possible. To Professor Matti Karjalainen I want to express my gratitude for supervising this work. For generic and specific advice and comments I also wish to thank Mr. Tero Tolonen, Mr. Cumhur Erkut, and Dr. Mikael Laurson. Professor Tapio 'Tassu' Takala I thank for the opportunity to work with sound synthesis models, and for new, inspiring ideas.

I have had the opportunity to work with great people under the umbrella of the DIVA project. Together we have had great time working with virtual acoustics, so thanks go to the whole bunch. Special thanks to Lauri Savioja and Tapio Lokki, who have shared the office with me, for good company and for an optimal signal-to-noise ratio.

The people at the Polytech Orchestra and the University of Helsinki Wind Orchestra I thank for the opportunities to study sound production of musical instruments, to explore rehearsal room and concert hall acoustics, and to practice my organizational skills in real world rather than in a virtual one.

Most of all, I want to thank my family for the endless support. My parents Leena and Vesa Hiipakka have always encouraged me during my education. My beloved wife Hanna I thank for the support, understanding, and love before, during, and after this work.

Otaniemi, Espoo, October 19, 1999 Jarmo Hiipakka

Table of Contents

List of Symbols

a	symbol for right-hand ring finger
$A(\omega)$	allpass filter simulating dispersion
$A(z)$	allpass filter transfer function
a_1	loop filter parameter
$b(n)$	impulse response of the guitar body model
$B(z)$	transfer function of the guitar body model
$\mathbf{C}$	coupling matrix
c	speed of the transversal vibration
D	fractional delay value
$d(n)$	dynamic delay parameter
$D(z)$	allpass filter denominator polynomial
e	symbol for right-hand little finger
$e(n)$	impulse response of the excitation model
$E(z)$	transfer function of the excitation model
$F(z)$	transverse force at the bridge
f_0	fundamental frequency
f_{n0}	body resonator center frequency on normalized frequency scale
f_S	sampling frequency
G	a function that maps the delay line signal to a delay parameter
g	loop filter zero-frequency gain
$G(\omega)$	frequency-dependent gain factor
$h(n)$	discrete-time impulse response
$H(z)$	transfer function in the z-domain
$H_l(z)$	loop filter transfer function
i	symbol for right-hand index finger
$I(z)$	filter approximation of the time-domain integration operation
m	symbol for right-hand middle finger
m_o	dual-polarization string output mixing coefficient
m_p	dual-polarization string input mixing coefficient
n	discrete time index
N	filter order
N_a	transient eliminator advance time
n_c	filter coefficient change time
N_P	effective length of a filter impulse response
p	symbol for right-hand thumb
$R_b(z)$	reflection function at the bridge in the z-domain
$R_f(z)$	reflection function at the fret in the z-domain
$s(n)$	string model impulse response
$S(z)$	string transfer function in the z-domain
T	string tension
T_d	dynamic string tension
$X(z)$	signal in the z-domain

$Z(z)$	bridge impedance
z^{-L}	delay of L samples
Δf_n	body resonator bandwidth on normalized frequency scale
$\delta(n)$	unit impulse
ε	a small positive number
ρ	linear mass density
ω	radian frequency

List of Abbreviations

CISC	Complex Instruction Set Computer
CWS	Commuted Waveguide Synthesis
FD	Fractional Delay
FIFO	First In, First Out
FIR	Finite Impulse Response
IIR	Infinite Impulse Response
MAC	Multiply-Accumulate instruction
MIDI	Musical Instrument Digital Interface
MPDL	Music Parameter Description Language
MPEG	Moving Picture Experts Group
NTP	Network Time Protocol
OSC	Open SoundControl
RISC	Reduced Instruction Set Computer
RRS	Recursive Running Sum
SAOL	Structured Audio Orchestra Language
SASL	Structured Audio Score Language
SDL	Single Delay Loop
SKINI	Synthesis toolKit Instrument Network Interface
TVFD	Time-Varying Fractional Delay
ZZ	Zetterberg–Zhang method

1 Introduction

The implementation and control of a model-based guitar synthesizer are described in this master's thesis. The sound of an acoustic guitar is simulated using a real-time implementation of a physics-based guitar model. The real-time model is implemented on a general purpose workstation platform using a high-level object-oriented programming style. For the control of the synthesis model, an attempt is made to qualitatively relate some of the playing techniques of the classical guitar to the parameters of the synthesis model.

A computer can generate any sound within the limits posed by sampling rate and resolution. Musically meaningful sounds can be created using a number of synthesis algorithms, which Smith (1991) divides into four categories:

1. abstract algorithms,
2. processed recordings,
3. spectral models,
4. physical models.

Abstract sound synthesis algorithms try to model neither a sound source nor a sound signal, but are mathematical methods used to create sound signals with varying temporal and spectral characteristics. In processed recording synthesis methods, recorded sound segments are combined and processed using various techniques. Examples of this approach are sampling and granular synthesis. Spectral modeling techniques are based on modeling the sound waves as they are perceived by the listener, whereas physical modeling is based on the sound source instead. For an overview and an evaluation of several different digital sound synthesis methods, the reader is referred to Tolonen *et al.* (1998a).

1.1 Physical Modeling and Model-Based Sound Synthesis

Physical modeling of musical instruments is an exciting paradigm in digital sound synthesis, and it has been a very active research field since the 80s. The basic idea behind the physical modeling approach is that when the vibrating structure is modeled with sufficient accuracy, the resulting sound will automatically be identical with the real sound of the modeled physical structure. The terms *physics-based sound synthesis* and *model-based sound synthesis* are also used to refer to modeling of sound sources.

In speech synthesis, the modeling approach has been used from the beginning of the 1960s, but earlier music synthesis techniques very little considered the sound source. Former synthesis methods try to imitate the properties of the sound signal, whereas physical modeling sets the focus to the imitation of the sound source itself.

Although the physical model parameters might not be directly measurable from the modeled instrument, they nevertheless are physically meaningful, and this is one of the greatest advantages of the physical modeling approach. Also perceptually important parts of the instrument tones' evolution will be automatically generated in a correct way.

A vast majority of the sounds possible to digital sound synthesis is musically uninteresting. If sound synthesis algorithms are designed without any reference to natural, acoustic sounds, it is easy to produce sound with unnatural and unpleasant character. One of the properties of acoustic sounds is the low-pass tendency of the tones, which is due to the stronger losses at higher frequencies. If the parameters of a physical instrument model are slightly modified from their original, "correct" values, it is possible to create sounds that are strange, and yet have a natural and familiar identity.

Välimäki and Takala (1996) divide the physical modeling techniques into five categories:

1. source-filter modeling,
2. numerical solution of partial differential equations,
3. vibrating mass-spring networks,
4. modal synthesis,
5. digital waveguide synthesis.

In this thesis the emphasis is on digital waveguide modeling, which is the most important and widely used physical modeling method both in academic and commercial applications.

1.2 Synthesis Model Control

It is important that a digital sound synthesis method can be controlled much the same way as traditional musical instruments are played, if it is to be used in musical applications of high quality. Currently, synthesizers are quite often criticized for their lack of expressive means. For a performing musician, the control interface must provide rather direct control of the synthesis parameters, and for automated music synthesis also the expression synthesis may be automated.

The MIDI protocol has enabled widespread usage of sound synthesizers in various applications. However, it may be that the keyboard-oriented MIDI protocol has radically limited the development of truly expressive synthesis methods. The prevalence of MIDI has forced all synthesizers to follow its limited conventions.

The control of a synthesis model may be divided roughly into three different levels, namely:

1. high-level control,
2. mapping of high-level control to implementation level,
3. implementation-level control.

The highest synthesizer control level consists of describing the desired properties of the synthesizer output using mostly musical terms and parameters, such as pitch, loudness, and articulation. The second level transforms the musical parameters into computational parameters that may be used by the synthesis algorithm. The lowest-level control includes implementing the synthesis algorithms carefully, so that unwanted transients and clicks are minimized. The implementation of the three control levels may be divided into separately running computing units as desired. For communication between such units, a communication protocol is needed. This thesis qualitatively describes some ways to map parameters from musical level to the lowest, signal processing level. However, the main emphasis is on the implementation of the lowest level control and on the communication protocol used for sound synthesis control.

1.3 Thesis Outline

Chapter 2 of this thesis describes the acoustic guitar and the related playing and expressive techniques, placing the main emphasis on the classical guitar playing techniques. In chapter 3 the development of the physics-based guitar model is described, and implementation of the model is presented in chapter 4. The parametrization and the real-time control problems of the model are described, and a new control protocol is introduced in chapter 5. Chapter 6 concludes the thesis.

2 Acoustic Guitar and Its Playing

The origins of the guitar are on the Iberian peninsula, which was later to become Spain. Nevertheless, the earliest references to the instrument are from fifteenth-century Italian sources (Tyler, 1980). The Italian term *viola* was used as a generic term to refer to any stringed instrument, but it was also used to specifically refer to a guitar-like instrument, whether or not the words *da mano* were added; the same is true for the Spanish term *vihuela de mano*, or *vihuela* (Figure 2.1). The terms *guitarra, chitarra*, etc. cannot positively be taken to mean the guitar until the sixteenth century (Tyler, 1980).

The guitar evolved from the 16th-century, four-course (double-stringed) *gittern*[1] to the five-course guitar of the barock period. The first guitars with six courses of strings were made ca. 1760–70 (Giertz, 1979). At the end of the 18th century the double courses were abandoned by many of the French guitar-makers in favor of single strings (Grunfeld, 1969). The Spanish luthier Antonio de Torres (1817–1892) perfected and codified the design and construction principles found in modern classical guitars by establishing the size and the proportions of the guitar body, as well as the fingerboard, and standardized the length of the strings (Figure 2.2).

The acoustic guitar is currently a very well-known and popular instrument used in many different musical contexts and styles. Nylon strings are used in guitars of classical and flamenco styles, and guitars with steel strings are common in folk, blues and jazz music. Guitars differ also in other respects than the string material, resulting a great variety of guitars with differing sounds. The basic structure of the acoustic guitar is nevertheless the same for each construction.

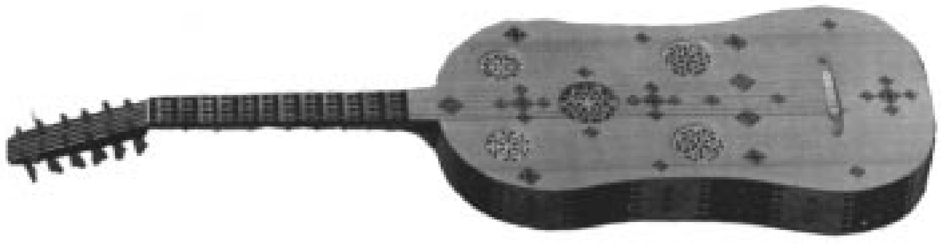

Figure 2.1: Reconstruction of the vihuela, c.1500, Spain. (Tyler, 1980; Plate 3c)

1. Not to be confused with the earlier instrument of the same name which, in fact, is a small treble lute.

Figure 2.2: Guitar by Antonio de Torres, Seville, 1883 (Grunfeld, 1969; Plate 227).

In this section the acoustic guitar is described mainly from the musicians perspective. The guitar and its playing is conceptually divided into three subparts, namely,

1. plucking the string,
2. string vibration, sustaining and damping it,
3. the guitar body.

Each of these parts has its own section in the following discussion, but first the basic construction of the instrument is briefly described.

2.1 Guitar Construction

The acoustic guitar is mainly made of wood. For quality instruments the quality and the processing of the timber is thus essential. The lumber is usually quarter-sawn and must be in moisture balance with the surrounding atmosphere (Cumpiano and Natelson, 1993). Guitars are usually made either of *seasoned* or *kiln-dried* wood. Seasoning is the process in which the timber is stored in controlled temperature and humidity conditions for a long period of time. Properly seasoned timber has lost most of its moisture during the process. In kiln-drying, the wood is artificially aged in a much faster procedure. Some luthiers think, however, that for the wood to be suitable for instrument-making, it must always be exposed to the atmosphere over a significant amount of time (Oribe, 1985).

The anatomy of a classical guitar is illustrated in figure 2.3. The guitar body consists of the top and the back plates and the ribs (sides), connected together with linings. The neck is connected to the body, and the fingerboard is in turn attached to the neck. The strings are attached to the bridge on the top plate of the guitar body, and the other ends of the strings are connected to the tuning machine in the guitar head. The guitar is assembled using different types of glues. The traditional choice is to use organic animal glue, but currently modern synthetic glues are commonly used.

2.1.1 Body Construction

The top plate or the soundboard of the guitar is the most important part determining the sound quality and character of the instrument, and is manufactured of two book-

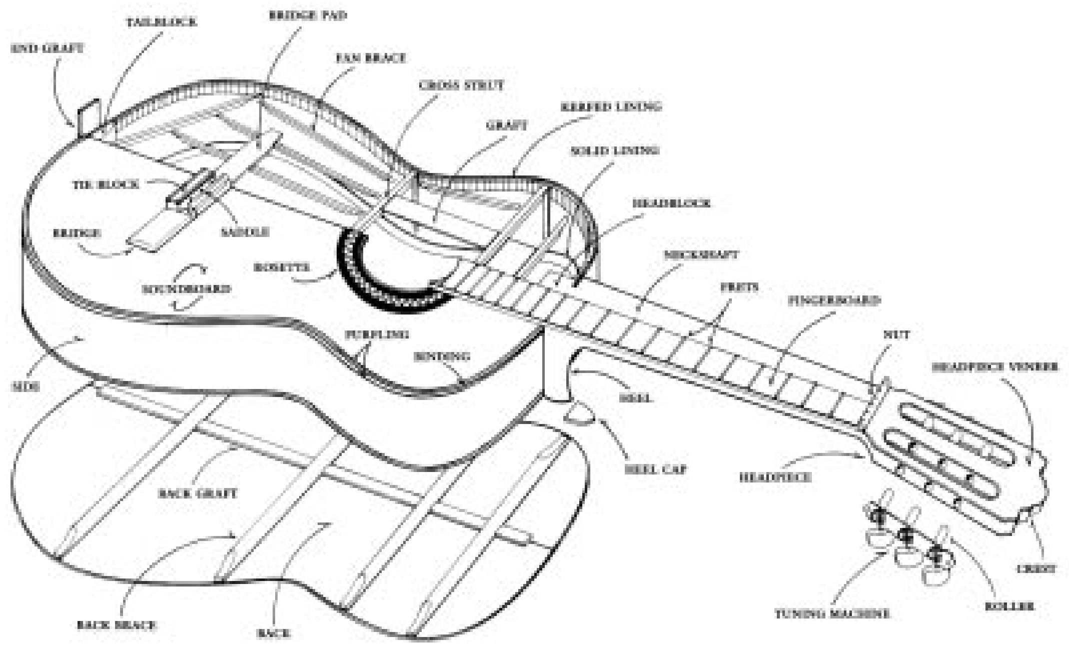

Figure 2.3: Exploded view of the classical guitar (Cumpiano and Natelson, 1993; Plate 1–2).

matched pieces of carefully selected timber for quality instruments. The first choice for the soundboard wood has traditionally been the close-grained German spruce, but due to the short supply of mature trees, also Sitka spruce, Eastern white spruce, as well as variety of cedars and redwoods are used (Cumpiano and Natelson, 1993). There are few direct comparisons of the soundboard materials, but, e.g., Elejabarrieta and Ezcurra (1997) have concluded that cedar would be a better material than the spruce. In the flamenco guitars, the surface of the soundboard is protected with *golpeadores* or *tapping-plates* from the fingernails of the right hand.

In the ribs and the back plate, the timber is less critical than in the top plate. Most frequently rosewood, mahogany, and maple are seen in modern guitars. In the finest classical models the sides and the back are made of palo santo (also called Brazilian or Rio rosewood, or jacaranda). In flamenco guitars, light-colored Spanish or Italian cypress is traditionally used.

To increase the strength of the construction and to spread the vibrations, braces (struts) are installed both to the top and the back plate. Several traditional designs for the top plate bracing patterns exist (figure 2.4). The pattern has a remarkable effect on the sound quality of the instrument and in every pattern there are a great number of variables. In the back plate the strutting pattern is usually quite simple. The material used for the struts is often the same as that used for the plate itself, however, even carbon fibre is sometimes used (Redgate, 1998).

The vibrations of the strings are radiated as sound waves through the vibrations of the guitar body. The strings are connected to the body via the bridge and the saddle, and vibrating strings apply a driving force to the body. The plates and the air cavity of the body provide a better impedance matching to the surrounding air than do the strings. In addition to the amplifying function, the body also colors the sound of the instrument

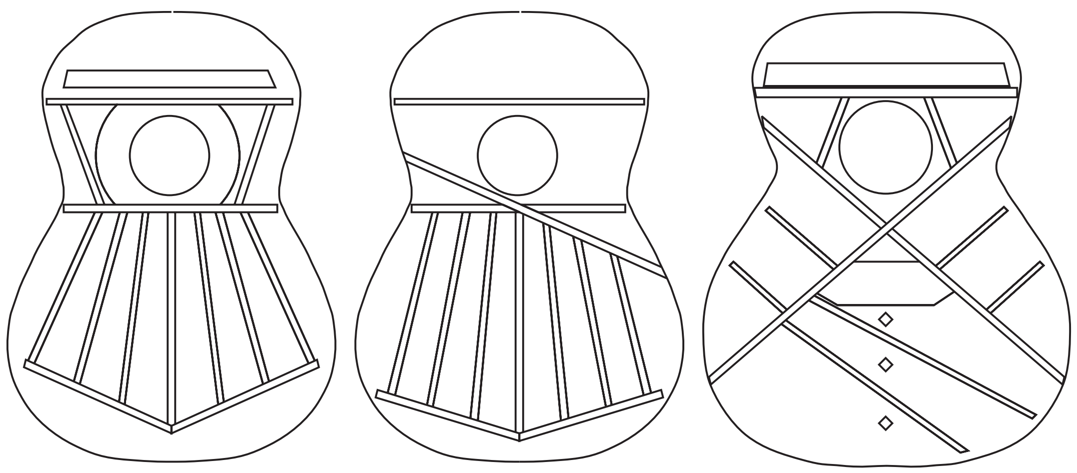

Figure 2.4: Bracing patterns: Torres pattern, asymmetric variation of the Torres pattern and X-bracing pattern for steel-string guitar (after Cumpiano and Natelson, 1993).

by amplifying some frequency components more than others and resulting in a specific sound quality for the specific instrument. The directional radiation pattern of the guitar is also determined by the properties of the instrument’s body.

The lowest body resonance of the guitar is typically in the range from 90 to 100 Hz and the second one is between 170 and 250 Hz. The lowest resonance corresponds to the first mode of the air cavity or the Helmholtz mode of the instrument body, and the second resonance is the first vibrational mode of the soundboard. The first few body resonances have fairly narrow bandwidths and thus also long decay times (Fletcher and Rossing, 1991).

The subjective sound quality of the guitar in respect to the acoustical properties of the guitar body have been studied by Meyer (1983a). He found that the third resonance of the guitar body around 400 Hz is of great importance to the overall tone of the instrument. The other criteria having great correlations with the positive subjective evaluations are the average levels of the frequency ranges approximately from 80 Hz to 1250 Hz. An attempt to relate these acoustical criteria to a few possible design parameters of the guitar soundboard construction is made in (Meyer, 1983b).

2.1.2 Neck and Fingerboard

Different types of acoustic guitars differ considerably in the construction and the design of the instrument’s neck. The neck is usually made of hardwood, such as mahogany. The neck can be made of a single piece of wood, but a more common construction is the three-piece neck, in which the head and the foot are made from separate pieces, the foot often being laminated. To support the neck against the bending force exerted by the strings, the steel-string guitars usually have a metallic *truss rod* under the fingerboard. The nylon-stringed classical and flamenco guitars have solid wooden necks.

The frets are mounted to the fingerboard or fretboard, which is traditionally made of ebony, but other hardwoods are also commonly used. In the classical and flamenco guitars, the fingerboard is flat, but in steel-stringed instruments they are often curved. Also the widths of the fingerboards differ radically between different guitar types, the classical and flamenco guitars having much wider fretboards than the steel-string guitars. The frets are metallic and are placed in positions resulting in an equal temperament tuning. Because the tension of the string increases when pushed against the fingerboard, the distance from the nut to the saddle is slightly greater than the scale-length would implicate. This increase is called string compensation (Fletcher and Rossing, 1991).

2.1.3 Strings and Bridge

Modern acoustic guitar strings can be divided into two categories: nylon and steel strings. In nylon-string guitars the three bass strings, i.e. 6th, 5th, and 4th, are made of nylon floss, with fine metal wire wound on the nylon core. The 3rd string is of single nylon filament or alternatively nylon floss wound with nylon filament. The 2nd and 1st strings are usually of plain nylon filament. The steel strings are made similarly of plain or wound steel wires. The tuning of the strings is adjusted with a tuning machine or with traditional tuning pegs at the head of the guitar.

The bridge is the anchoring point of the strings on the soundboard. It determines the spacing of the strings as well as the height of the strings above the fretboard. The bridge couples the vibrations and distributes the tension of the strings onto the soundboard. The design and the construction of the bridge may have a dramatic effect on the sound quality and the action of the instrument. The standard modern design of the classical guitar bridge (see Figure 2.5) with rectangular wings, a tie block and a removable bone saddle was created by Antonio de Torres around 1850 (Cumpiano and Natelson, 1993). Meyer (1984b) has studied the influence of different bridge designs on the quality of the instrument tone, and found that a bridge being shorter and wider than the standard may, in fact, be advantageous.

The forces that the strings exert on the bridge vary according to the polarization of the string vibrations. The vibrations parallel and perpendicular to the soundboard excite different sets of guitar body resonances and the decay rate of the vibratory motion also depends on the angle through which the string is plucked. When the string is plucked perpendicular to the soundboard, a strong but rapidly decaying tone will result. When the string is plucked parallel to the top plate, a weaker but longer tone is obtained (Taylor, 1978).

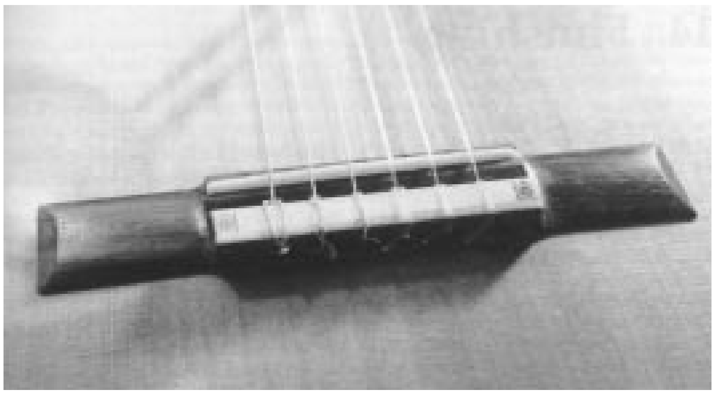
Figure 2.5: Classical guitar bridge (Middleton, 1997; pp. 123)

2.2 Plucking the String

The acoustic guitar is excited by plucking the strings either using fingertips or with a plectrum. This thesis will concentrate on the finger-picking playing styles. In classical guitar playing, two kinds of strokes are used to excite the string: in *apoyando* (Spanish for ‘leaning on’) strokes the finger comes to rest on the next string after the string has been excited, in *tirando* (‘pulling’) the adjacent string is not touched. Other plucking techniques include, e.g., the *rasgueado* techniques, which are mainly used in flamenco music.

One important subject concerning the right-hand techniques of guitar playing is the use or non-use of nails. Fernando Sor (1778–1839), the greatest guitarist of the romantic era, and the writer of ‘Méthode pour la guitare’ (1830), did not use fingernails, but only the fleshy part of the fingertips. His contemporary and friend, the Spanish virtuoso Dionisio Aguado (1784–1849) was firmly pro-nail. The issue has actually never been solved; the flamenco guitarists use their nails, whereas classical guitarists try to make best of both worlds by plucking the strings using fingertips in conjunction with carefully shaped nails (Grunfeld, 1969). Today guitar players typically shape their nails so that the string makes a brushing contact with the flesh. This way the “click”, that can result from the contact between the vibrating string and the nail, is reduced. A very thorough discussion of the advantages of finger nail usage and the function of the nails is presented in (Taylor, 1978).

The right-hand fingers have conventional symbols widely used in guitar music and literature. The symbols are:

p	for thumb (‘pulgar’),
i	for index or 1st finger (‘indice’),
m	for middle or 2nd finger (‘medio’),
a	for ring or 3rd finger (‘añular’),
e	for little or 4th finger (‘meñique’).

The little finger is not used in the orthodox classical guitar playing[1], and the symbol **e** is not as established as the others (letter **c** is also used for the little finger).

Several phenomena take place, when a string is plucked. If the string to be plucked is already vibrating, the finger first quickly damps the original vibration at the same time starting new vibrations on both sides of the plucking position. The finger pushes the string down toward the soundboard, and then the string is allowed to slip over the fingernail, setting the string to vibrate.

The tone character can be controlled by varying the plucking position and plucking style. Basic techniques of plucking the string are presented in the following subsections. These styles and the variable parameters give the performer a great repertoire of tone character.

1. The little finger was earlier used to balance the guitar and to steady the hand by placing the finger on the top of the guitar or against the bridge.

2.2.1 Apoyando and Tirando Strokes

In the *apoyando* or *rest stroke* the plucked string is pulled towards the soundboard, where it is allowed to slip over the fingernail. The finger continues its movement until it comes to rest on the adjacent string. The nail acts as a ramp, which converts some of the horizontal finger motion into vertical motion of the string (Fletcher and Rossing, 1991).

In the *tirando* or *free stroke*, the line of impact is parallel to the soundboard of the guitar rather than down towards it as in apoyando, and the striking finger does not touch the adjacent string. The finger starts from a short distance from the string, where from it accelerates to strike the string with the tip of the nail and stops after a follow-through. When the striking finger touches the string, the tip-joint of the finger slightly bends backwards, if the finger is fully relaxed. The player can control the amount of tension with respect to the musical context.

According to Pavlidou and Richardson (1997), the main difference between the apoyando and tirando strokes is the angle with which the string is released. The almost horizontal finger movement of the tirando stroke forces the string to move nearly parallel to the instrument soundboard, and results in a quieter sound of longer duration than the apoyando stroke. Fletcher and Rossing (1991), however, see little difference between the strokes in this regard, but conclude that the player may alter the balance between horizontal and vertical string motion by changing the angle of his fingertip.

It is usually possible to play louder using an apoyando stroke, and it may be used to discern the melody line from a dense accompaniment. The usage of the rest stroke is, however, naturally limited by the fact, that leaning on the adjacent string will of course damp its vibration. Therefore, some writers think that the player must develop his free stroke to produce notes equivalent to those played with rest stroke (Russell, 1988). Some tutors aim for core sound that would be the same for the rest stroke and the free stroke, yet keeping the tonal extremes of the techniques available for contrast (Duncan, 1980).

The differences between the apoyando and tirando strokes may be very individual to each player. Beginners typically have big differences in the sound quality according to the playing technique, whereas advanced players are capable of very subtle intentional variations and even sound quality irrespective of the plucking style (Helminen, 1999).

2.2.2 Rasgueado Techniques

The *rasqueado* techniques give the characteristic sound to the flamenco guitar music, and it is the flamenco music that has influenced modern composers so that they have included rasgueado techniques in their work (Duncan, 1980). The term rasgueado (or alternatively *rasgueo*) embraces all strumming techniques using one or more fingers of the right hand. The techniques range from single index finger strokes to the longer and more advanced rolling sequences (Campbell, 1978).

In the simplest, index finger rasgueado, two kinds of strokes are used: downstrokes and upstrokes. In downstroke the index finger is first flexed from the knuckle, so that the finger almost touches the base of the right-hand thumb. From this position, the finger is flicked forward, the nail striking downwards across all the six strings of the guitar. During the upstroke, the finger is flicked back towards the starting position. The finger now sounds the higher-pitched strings first. Generally, fewer strings are plucked during the upstroke than during the corresponding downstroke.

The four-stroke rasgueado is a very important technique in flamenco playing. It consists of a rapid succession of four downstrokes by the right-hand fingers, in the order **e**, **m**, **a**, **i**. The accent of the four-stroke rasgueado is on the fourth stroke, i.e. on the index finger. The accent also coincides the beat of the music.

The five-stroke rasgueado has two differences compared to the four-stroke rasgueado. The accent of the rasgueado coinciding with the beat of the music falls on the first stroke made by the little finger. The second difference is that the rasgueado ends with an index finger upstroke after the **e**, **a**, **m**, **i** sequence of downstrokes. Continuous roll may be achieved by curling the other fingers to the palm when the index finger extends for its downstroke. This way the fingers will be ready for the repetition of the five-stroke rasgueado by the time the index finger has made the last upstroke.

Several other types of rasgueados can be described, all consisting of series of up- and downstrokes with right-hand fingers and exciting more than one string in rapid succession.

2.2.3 Pluck Position

The effect of the plucking position is to produce a comb-filtering effect on the resulting sound. When the string is plucked at the middle of the string, the amplitude distribution of all the even-numbered harmonics is zero. In general, a vibrational mode of an ideal string will not vibrate, if the string is excited at one of the nodes of the mode (Fletcher and Rossing, 1991). For example, if the string is plucked 1/5 of the distance from one end, the spectrum of mode amplitudes will be like shown in figure 2.6. Note the absence of the 5th, 10th, etc. harmonic. In a non-ideal string, however, the modes of the string vibration are in general nonlinearly coupled so that a mode with zero initial energy will begin to vibrate (Legge and Fletcher, 1984). Also the finger or the plectrum exciting the string always is of finite width, and thus there will be no mode with zero vibration.

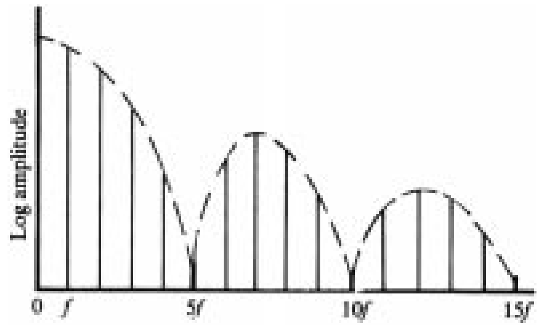

Figure 2.6: Spectrum of a string plucked one-fifth of the distance from one end (Fletcher and Rossing, 1991; Fig. 2.7).

The pluck position is used to vary the tone color of the guitar sound. The neutral position of the right hand is just behind the sound hole. Playing *sul tasto* (near or over the fingerboard) produces mellow tone quality whereas *sul ponticello* (close to the bridge) gives a brighter and more metallic tone than the neutral pluck position. This effect can be used for major changes in tonal color. Nuances of color are produced by changing the angle with which the string is released, and the angle of the fingernail against the string direction (Duncan, 1980).

2.3 String Vibration

The vibratory motion of the guitar string consists of three components, namely transversal, longitudinal, and torsional vibrations. The transversal vibration is further divided into horizontal and vertical components, which correspond to the vibrations along and perpendicular to the top plate of the instrument, respectively. The longitudinal and torsional vibrations can be considered non-essential in synthesizing the sound of acoustic guitar, and are not considered in this work. Torsional waves may, however, be important in the plucking process (Pavlidou and Richardson, 1997).

The player affects the vibration of an excited string by controlling the pitch of the sound with his left-hand fingers. Also the damping of the string vibrations can be controlled using either left or right hand.

2.3.1 Vibratory Motion

The transversal vibratory motion of the string can be expressed as a sum of modes that are obtained by solving the wave equation for the string. The equation for transverse waves in a lossless string is

$$\frac{\partial^2 y}{\partial t^2} = \frac{T}{\rho}\frac{\partial^2 y}{\partial x^2} \tag{2.1}$$

where T is the tension of the string, ρ is the string's linear density, y is the displacement of the string, x is the distance from the origin along the string, and t is time. The general solution of the wave equation can be described as two traveling waves traversing the string in opposite directions and reflecting back from the ends of the string. A string with fixed ends has harmonic normal modes of vibration and the general solution can be described as a sum of normal modes (Fletcher and Rossing, 1991).

In a real string the vibratory motion attenuates gradually because of the external and internal losses and as a result of transmission of the vibrations to the guitar body. When all the energy in the string is consumed, the string comes to rest. In the low-frequency range the attenuation is mainly caused by the air resistance and the losses at the string terminations (Chaigne, 1992). For the higher frequencies the internal losses of the string material become essential (Chaigne, 1991). Because of the small diameter of the strings, the radiation of the energy to the air is usually negligible (Chaigne, 1992).

The harmonics in a real string are not exactly in integral ratios, because the finite stiffness of the string causes the velocity of the vibratory motion to be frequency-dependent. This effect is called dispersion.

The stretched string is linear only to the first approximation since the displacement of the string causes a second-order change in the length, and thus the tension, of the string (Legge and Fletcher, 1984). Therefore the calculated mode frequencies are not exact for a vibrating string. The result of the change in tension can be heard as a change in the fundamental frequencies of louder guitar tones. The coupling between the bridge resonance and the string is another important cause for non-harmonicity of the partials of the guitar tones.

The string vibrations in an acoustic guitar are connected to the other strings of the instrument through the mechanical coupling and via the air. This is called sympathetic coupling.

2.3.2 Horizontal and Vertical Polarizations of String Vibration

The transversal vibrations of the guitar string can be divided into two polarization components: the horizontal and the vertical polarization, i.e., vibrations parallel and perpendicular to the soundboard, respectively. As described in section 2.1.3, the bridge characteristics differs in these two directions. The difference leads to slightly different vibration in the two directions.

Two important phenomena resulting from the two polarizations are the two-stage decay rate of guitar sounds and the modulation of partials of a guitar tone. The former effect is a result of different decay rates of the two vibration polarizations (Fletcher and Rossing, 1991) and the latter suggests that the frequencies of the two polarization components are not the same. The similar two vibration polarizations in piano strings have been described by Weinreich (1977).

2.3.3 Controlling the Pitch

The pitch of the guitar is controlled by the player mainly using the left-hand fingers. The player can change the length of the vibrating part of the string by pressing the string down to contact the frets of the fingerboard. The string is normally stopped as close to the fret as possible in order to get clean, non-buzzing tones with minimum effort. During the history of the guitar, the left hand technique has been of little controversy; the early sources given practically same the principles of left-hand fingering as found on modern guitar tutors (Tyler, 1980).

In guitar notation, the left-hand fingers are numbered as follows:

1	for index finger
2	for middle finger
3	for ring finger
4	for little finger

The left-hand thumb is not normally used in fingerings, but is placed opposite the fingers, and balances the pressure of the other four fingers

The term *ligado* (Spanish for 'tied') refers to sequences of notes sounded by the left hand only, with a plucked excitation by a right-hand finger only to the first note of the sequence. In 'hammering on' ligado a left-hand finger descends firmly (with a hammer-like action) onto an already vibrating string to produce a note of higher pitch. 'Pulling off' sounds a note of lower pitch by plucking the string with a left-hand finger stopping the string on a fret. The note produced by pulling-off may be stopped by another left-hand finger, which may again be 'pulled off' to sound yet a lower note.

In classical guitar technique, the left-hand fingers are normally used to stop only one string behind a fret. Sometimes the index finger is however placed straight across all six strings, stopping all strings behind the same fret. This technique is called *barré* or *bar*. Barré allows playing of chords in various positions along the fretboard. Besides the full barré stopping all the six strings, the index finger may be used to stop fewer strings in five- or four-string barré or two or three strings in a true half-barré. The tip segment may also be used to cover lower strings without recourse to the full barré (Duncan, 1980).

In *glissando*, the string is struck once to sound a note of certain pitch. The pressure of the left-hand finger is maintained and the finger is rapidly slid up or down the string, so that the arrival at the new fret produces the desired pitch. The extent to which the intervening chromatic series of notes is sounded may be varied.

Vibrato is produced by rapidly changing the tension of the string; when the tension is increased, the pitch of the note is sharpened, and when the tension is decreased, the pitch will decrease as well. Vibrato can be produced either longitudinally or laterally to the string. In longitudinal vibrato, the string is literally stretched in alternate directions, so that the pitch is both increased and decreased from the mean tone. On positions, where this technique is inefficient, lateral-bend vibrato can be used. In this form the finger, flexing from its tip and middle joints, moves the string from side to side. Since this method can only increase the tension of the string, the pitch will only fluctuate upwards.

The *capo* (or *capo tasto*) is a device which is fixed across the strings to shorten the string length, thus resulting in upward transposition without altered fingering. It acts as a movable nut, which stops the strings when tightly fixed behind a fret. The Spanish capo, *cejilla* is widely used by flamenco guitarists to give added brightness to the performance. Usually the capo influences the length of all the strings, but capos capable of stopping only selected strings have also be manufactured.

2.3.4 Harmonics

Harmonics or *flageolets* refer to the pure bell-like tones produced by sounding the whole string in fractions of its length. Natural harmonics are the tones from the harmonic series of the open strings. They are made by forcing a node on a specific location along the string by lightly touching the string over an appropriate fret with a left-hand finger and then lifting the finger away immediately after the right-hand excitation. Because it is of no use to touch the string on one nodal point, while plucking at another, plucking positions rather near the bridge are chosen to obtain a strong harmonic.

If the string is pressed against any of the frets, a complete harmonic series will naturally be accessible relative to the new string length. To produce these *artificial harmonics*, the string must be both plucked and touched lightly using right-hand fingers, so that the left hand is left free to press the string down on any desired fret. The index finger is used to touch the string and the thumb or the ring finger is used to actually pluck the string. Because the plucking position cannot be very far from the index finger touching the string, and thus is not near the bridge, the artificial harmonics tend to sound weak.

2.3.5 Damping the String Vibration

The player has many alternative ways of stopping the vibrations of the string. In principle any fleshy part of either hand can be used to damp the strings. The most effective technique is to stop the vibrations of the strings is by bringing the right hand onto the strings so that the edge of the palm contacts the strings.

Any right-hand finger may be used to damp the vibrations of any of the guitar strings. The most typical example is the apoyando stroke, in which the finger exciting a string comes to rest on the adjacent string effectively damping it.

The right hand may also be used to produce a *pizzicato* effect: if the hand is placed with the palm in contact with the strings near the bridge, when the string is plucked, frequency-selective damping is introduced to the string. The higher modes of string vibration, which have antinodes near the bridge, are strongly suppressed. This technique produces a muffled, “woolly” sound (Taylor, 1978).

To damp the strings, left-hand fingers can be used in two alternative ways. If a left-hand finger fretting a certain note is lifted from the fingerboard so that the contact with the string is still maintained, the string vibration will die out rather quickly. The second way is to bring, e.g., the little finger down lightly straight across the strings.

2.4 Guitar Body

The way of holding the guitar is mainly dictated by the musical style: classical guitar has its own playing position differing e.g. from the flamenco position. However, the posture of the player affects the sound of the instrument somewhat, because different playing positions have different implications on the vibrations of especially the back plate and the sides of the guitar body. The guitar body itself can be “played” by tapping the guitar with either the palm or the fingers of the player’s right hand.

2.4.1 Playing Positions

Although each player ultimately has a rather individual playing position, a few nominal positions can be described for different musical styles. In the classical guitar playing position the guitar is placed on the players left thigh. A footstool is placed under the left foot to incline the left thigh and to bring the back edge of the upper side of the guitar closer to the players chest. The traditional flamenco way of holding the guitar is to support the weight of the instrument on the right thigh so that the larger curve of the guitar body is placed on the outer side of the thigh. The right arm rests on top of the guitar, the weight of the arm balancing the guitar (Campbell, 1978).

Both the classical and the traditional flamenco playing positions have evolved so that the vibrations of the guitar body are only slightly affected by the player. The widespread use of the acoustic guitar in different musical styles has given raise for many different, individual playing positions. The hollow of the guitar body can, e.g., be rested on the right leg, which has been crossed over the left leg, or a neck strap can be used if guitar is to be played in a standing position.

2.4.2 Tapping the Guitar Body

The guitar body can be tapped from various places to achieve special effects. These percussive strokes are commonly used in contemporary guitar music. The most common places to strike are the bridge and different parts of the soundboard. To tap the body, player may use palm, fingers, fingertips, the side of the thumb or even the knuckles. Nails are seldom used, because they might be harmful to the finishing of the instrument. The best places to achieve desired percussive effects may vary from instrument to instrument, and experimentation is required when these effects are needed (Russel, 1988).

The *tambour* or *tambora* is a kettledrum effect, that includes striking the saddle of the guitar bridge with the right hand. The sound is a mixture of the strings and the body, and its exact properties are determined by the position of the stroke. A considerable amount of the energy of the stroke is transferred to the guitar bridge and body in a rather straightforward manner, so that the effect may be considered with the body-tapping effects.

The *golpe* (Spanish for ‘tap’) is often used in flamenco music and is made by tapping the guitar soundboard by the 3rd finger of the right hand, bringing both nail and flesh into contact with the guitar body. Flamenco guitars are therefore equipped with a tap-plate, which protects the instrument’s soundboard. The golpe is used both on its own and simultaneously with strokes to the strings (Campbell, 1978).

3 The Guitar Model

The acoustic guitar is an extensively studied and simulated instrument (Grunfeld, 1969; Christensen and Vistisen, 1980; Pavlidou and Richardson, 1997; Karjalainen *et al.*, 1998) and many different sound synthesis algorithms have been demonstrated using guitar sounds. This is partly because of the popularity of the instrument, partly because realistic guitar sounds are fairly easy to produce due to the fact that the string is to the first approximation a simple one-dimensional vibrating structure.

In the following discussion the development of a physics-based acoustic guitar model is described. This thesis closely follows the formulation given by Tolonen (1998) and Karjalainen *et al.* (1998). The purpose is to introduce the structure of the synthesis model; readers interested in calibration of this model to achieve realistic guitar sound synthesis are referred to Välimäki *et al. (*1996) and Välimäki and Tolonen (1998).

From the modeling point of view, guitar can be divided into two functionally separate sub-structures: the strings and the body. The third functional part of the model is the part corresponding to the excitation by plucking the string. These functional parts are depicted in figure 3.1. The subsections 3.1–3.3 present both simple and advanced string models based on digital waveguides. Body modeling is discussed in section 3.4, and model excitation by plucking in section 3.5. Discussion of computationally efficient multirate structures for guitar modeling in section 3.6 concludes this chapter.

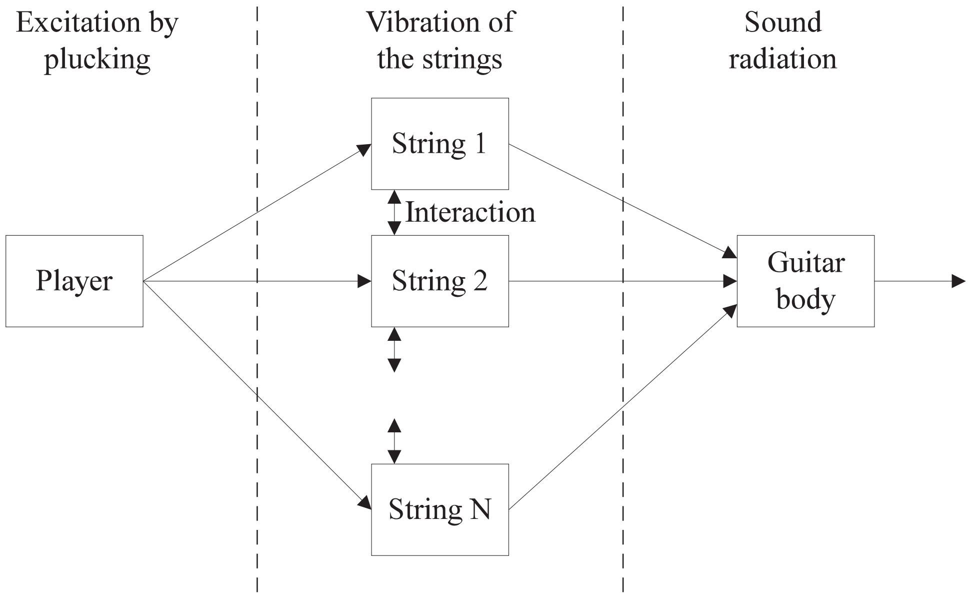

Figure 3.1: Block diagram of the sound production mechanism in the acoustic guitar, after (Karjalainen *et al.*, 1993).

The following discussion assumes that the reader is familiar with the concept of z-transformation of discrete-time systems. The z-transform for the discrete-time systems is quite similar to the Laplace transform for continuous-time systems. The relationship between the input and output of a discrete-time system involves multiplication by the appropriate z-transform, rather than convolution as for the signals themselves. Zeros and poles having the same role as for continuous-time systems may be defined using the z-transform, and the frequency response of the system can be derived from the transform and related to an appropriately defined Fourier transform. For more information on Laplace, Fourier, and z-transforms, the reader is referred to a standard textbook on digital signal processing such as (Oppenheim *et al.*, 1983).

3.1 Modeling of Vibrating Strings Using Digital Waveguides

The theory of digital waveguides has primarily been developed by Smith (1987, 1992, 1993) and the method was first applied to artificial reverberation (Smith 1985). The method suits particularly well to the simulation of one-dimensional resonators (vibrating strings, acoustic tubes, thin bars), which are found in many families of musical instruments. Waveguide techniques have also been applied to a variety of two- and three-dimensional acoustical problems (Van Duyne and Smith, 1993; Savioja *et al.*, 1994).

A well-known forerunner of the waveguide synthesis models is the Karplus-Strong algorithm developed by Kevin Karplus and Alex Strong for the synthesis of plucked string instrument sounds (Karplus and Strong, 1983). The original Karplus-Strong model has been further extended by Jaffe and Smith (1983), and has later led to a number of detailed models of string instruments (Karjalainen and Välimäki, 1993; Smith, 1993; Välimäki and Tolonen, 1998).

3.1.1 Digital Waveguide Model for a Lossless String

As stated in section 2.3.1, the general solution of the wave equation (Eq. 2.1) for a lossless flexible string can be interpreted as a linear combination of two traveling waves proceeding in opposite directions along the string. This is also known as the d'Alembert's solution. The digital version of the solution can be obtained by discretization of the functions representing the traveling waves.

The sampled traveling waves can be interpreted as delay lines depicted in figure 3.2 (Smith, 1992). The output of the waveguide model at point *k* is given by the sum of the two waveguide variables at that point. The digital waveguide provides exact solution to the wave equation at all integral points, if the original wavefronts are bandlimited to the Nyquist frequency. Values at fractional points along the waveguide may be obtained using *fractional delay filters* (see section 3.1.4).

It is worth noting that the input and output signals to the delay line of figure 3.2 may be of any wave variable type, such as displacement, velocity, acceleration or slope (Smith, 1992). Selecting acceleration as the wave variable has an interesting consequence since then an ideal pluck becomes an impulse (Karjalainen and Laine, 1991). The derivation in section 3.1.3 assumes that this choice has been made.

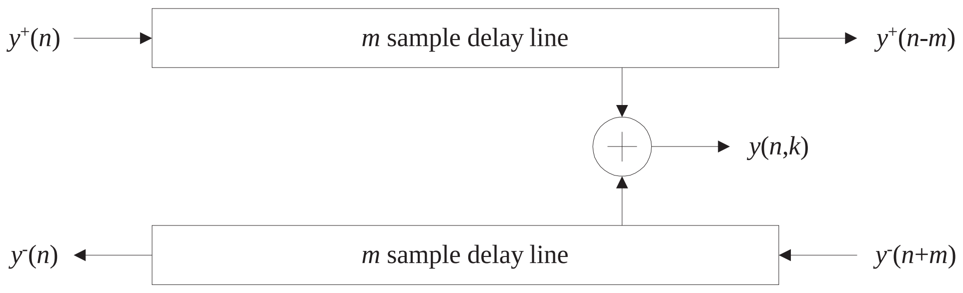

Figure 3.2: One-dimensional digital waveguide, after (Smith, 1992).

3.1.2 String Terminations, Damping, and Dispersion

The string terminations in the digital waveguide model can be represented by reflection filters, which produce phase inversion and frequency dependent damping. Due to the losses always present in a real string, the string vibration also attenuates gradually and the stiffness of the string causes dispersion.

The frequency-dependent losses in the string cause exponential attenuation of the wavefronts traversing the string. This corresponds to multiplication of the traveling waves by a frequency-dependent constant at each time step. The gain factors can be lumped together for every unobserved portion of the string, because the system is linear (Smith, 1987). This is illustrated in figure 3.3.

The finite stiffness of a real string causes the velocities of the traveling waves to be frequency-dependent, the high-frequency signal components having greater velocity than the lower frequencies. This effect is called dispersion. In a string model, dispersion can be taken into account by using allpass filters, that have group delays approximating the effect found in real strings. The necessary allpass filters are also depicted in figure 3.3.

3.1.3 Single Delay Loop Formulation

The above formulation of the bidirectional waveguide string model can be simplified to a string model with the loop consisting of a single delay line. This results in a single

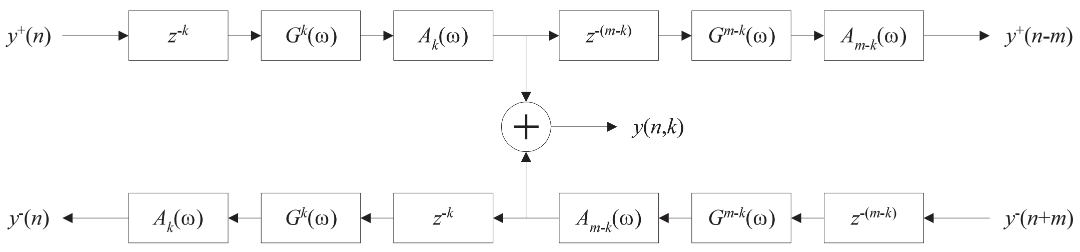

Figure 3.3: A digital waveguide with frequency-dependent gains $G(\omega)$ are lumped together between observation points. Dispersion is similarly implemented with allpass filters $A_l(\omega)$ that approximate the dispersion in a string section of length l.

delay loop (SDL) string model (Karjalainen *et al.*, 1998). The effects of the excitation and pickup (in case of electric guitars) positions can be easily simulated using the bidirectional waveguide model, but they may also be incorporated to the computationally more efficient SDL model (Karjalainen *et al.*, 1998). In the following discussion a string model with transversal bridge force as output is described, ignoring the effects resulting from different pickup microphone positions. The model is applicable to acoustic guitar sound synthesis. A more elaborate derivation of the single delay loop string model is given in (Karjalainen *et al.*, 1998).

The basis for the SDL model is the bidirectional waveguide model of figure 3.4. The notation $H_{A,B}(z)$ in the figure refers to the z-domain transfer function from point A to point B. The original excitation $X(z)$ has been divided into two parts, so that $X_1(z) = X_2(z) = X(z)/2$. The filters $R_f(z)$ and $R_b(z)$ represent the reflections at the fretboard and the bridge end of the string, respectively. The output of the string model is the transverse force $F(z)$ at the bridge. The force can be derived from the acceleration signal traversing the string by integration and multiplication with the bridge impedance $Z(z)$. The filter $I(z)$ represents the discrete-time approximation of the time-domain integration operation.

The model may be first simplified by introducing at point $E1$ a single excitation signal, that corresponds to the combined effect of the two excitations $X_1(z)$ and $X_2(z)$. The *equivalent excitation* is expressed as

$$\begin{aligned} X_{E1}(z) &= X_1(z) + H_{E2,L2}(z)\,R_f(z)\,H_{L1,E1}(z)\,X_2(z) \\ &= \frac{1}{2}\,[1 + H_{E2,E1}(z)]\;X(z) \qquad , \\ &= H_E(z)\;X(z) \end{aligned} \tag{3.1}$$

where

$$H_{E2,\,E1}(z) = H_{E2,\,L2}(z)R_f(z)H_{L1,\,E1}(z) \tag{3.2}$$

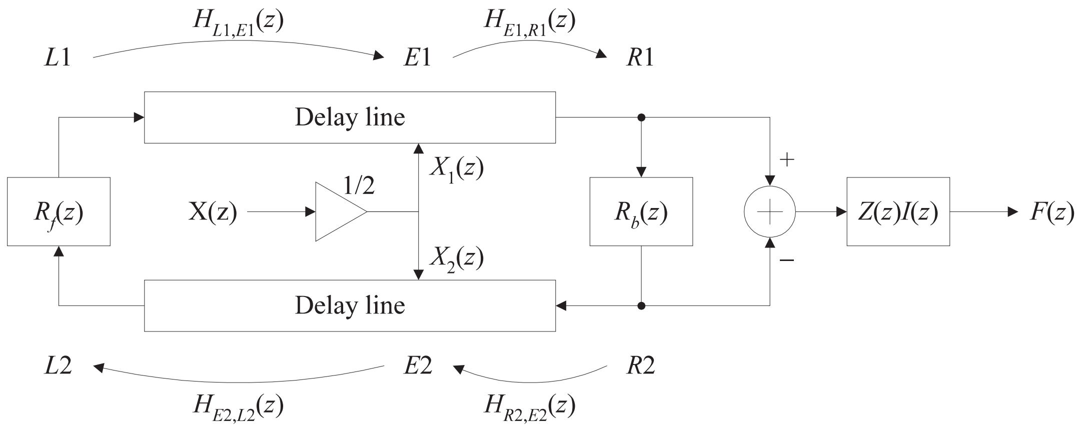

Figure 3.4: Dual delay line waveguide model for a plucked string with output at the bridge, after (Karjalainen *et al.*, 1998)

is the left-side transfer function from $E2$ to $E1$, that consists of the transfer functions from $E2$ to $L2$ and $L1$ to $E1$, and of the reflection function $R_f(z)$. The introduction of the equivalent excitation signal and linearity of the system allow us to combine all components of the string loop into a single transfer function

$$\begin{aligned} H_{loop}(z) &= R_b(z)H_{R2,E2}(z)H_{E2,L2}(z)R_z(z)H_{L1,E1}(z)H_{E1,R1}(z) \\ &= R_b(z)H_{R2,E2}(z)H_{E2,E1}(z)H_{E1,R1}(z) \end{aligned} . \qquad (3.3)$$

Now the string transfer function $S(z)$ that represents the recursion around the string loop, can be written as

$$S(z) = \frac{1}{1 - H_{loop}(z)} . \qquad (3.4)$$

The string loop can be implemented efficiently, if the pure delay and the frequency-dependent components of the transfer function $H_{loop}(z)$ are separated, so that

$$H_{loop}(z) = z^{-L}H_l(z) . \qquad (3.5)$$

Here it is assumed that the delay component can be represented with sufficient accuracy using an integer-length delay line z^{-L}. Otherwise, a fractional delay approximation must be used, as described in section 3.1.4.

A popular choice for the loop filter $H_l(z)$ in equation 3.5 is the one-pole filter (Jaffe and Smith, 1983; Tolonen, 1998)

$$H_l(z) = g\frac{1 + a_1}{1 + a_1 z^{-1}} . \qquad (3.6)$$

In equation 3.6, the parameter g determines the zero-frequency gain of the loop filter, and the parameter a_1 determines the cutoff frequency of the filter. The one-pole filter is the simpliest filter that can be used to implement slow attenuation of the lower frequencies and rapid attenuation of the higher frequencies. The formulation of the loop filter given in eq. 3.6 allows for separate adjusting of the DC attenuation and the frequency-dependent characteristics of the filter, and is therefore favorable over other other formulations of the same filter.

The overall transfer function for the string model from excitation to bridge force output is (using the definitions above)

$$H_{E,B}(z) = \frac{F(z)}{X(z)} = H_E(z)H_{E1,R1}(z)S(z)H_B(z) , \qquad (3.7)$$

where

$$H_B(z) = Z(z)I(z)[1 - R_b(z)] . \qquad (3.8)$$

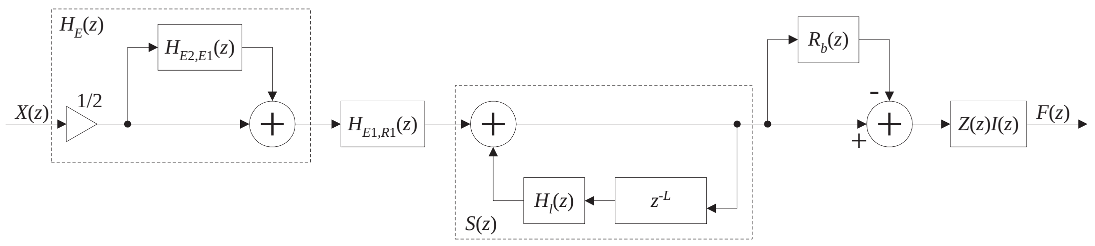

Figure 3.5: A SDL string model with equivalent excitation, single string loop, bridge impedance and integration. The model also takes into account the wave propagation from excitation point to bridge and the reflection characteristics of the bridge.

The model described by equation 3.7 is illustrated in figure 3.5.

The string model previously developed can be further simplified for practical sound synthesis as described in (Karjalainen *et al.*, 1998). The transfer function $H_{E2,E1}(z)$ is almost a lossless phase-inversive delay and the lowpass section may be replaced with a negative constant slightly less than 1 in absolute value. The losses in wave propagation from excitation point $E1$ to bridge position $R1$ are also negligible and the block $H_{E1,R1}(z)$ may be left out. The reflection function $R_b(z)$ is a nearly ideal, phase-inversive fixed end for the vibrating string, so that the term $[1 - R_b(z)]$ may be approximated by constant 2. The structure of the string loop $S(z)$ cannot be simplified, because the delay and the loop filter are critical to tone quality. A new timbre control filter is introduced into the model, so that errors due the reductions can be compensated for. The resulting model is presented in figure 3.6.

3.1.4 Fractional Delay Filters in String Models

The length L of the delay line inside the string loop approximately determines the fundamental frequency f_0 of the synthesized tone as

$$f_0 = \frac{f_S}{L} \tag{3.9}$$

where f_S is the sample rate of the model, and the length L is measured in samples. The pure digital delay line allows the length only to have an integral value, thus limiting the possible fundamental frequencies to discrete values. The error in the pitch of the

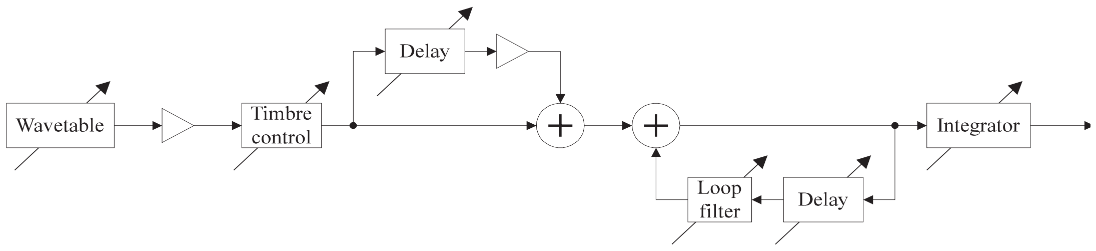

Figure 3.6: A string model for practical acoustic guitar sound synthesis. The model of figure 3.5 has been somewhat simplified, and a new timbre control filter has been added.

synthesized sound becomes larger as the fundamental frequency increases and is intolerable in musical applications (Jaffe and Smith, 1983).

The problem can be avoided and real-valued delay line lengths can be obtained using a *fractional delay* (FD) filter in each string loop. The two most attractive design methods for fractional delays in digital waveguide modeling are the classical *Lagrange interpolation* method for FIR (finite impulse response) filters and the *Thiran interpolator* for the IIR (infinite impulse response) filters. The main advantage of these methods is that they approximate ideal interpolation accurately at frequencies near the fundamental frequencies of speech and music signals (Välimäki, 1995).

The transfer function of a FIR filter is of the form

$$H(z) = \sum_{n=0}^{N} h(n) z^{-n}, \tag{3.10}$$

where N is the order of the filter, and $h(n)$ ($n = 0,1,\ldots,N$) are the coefficients (and the impulse response) of the filter. For an FIR structure that implements Lagrangian interpolation, the coefficients are

$$h(n) = \prod_{\substack{k=0 \\ k \neq n}}^{N} \frac{D-k}{n-k}, \qquad n = 0,1,2,\ldots,N, \tag{3.11}$$

where D is the desired fractional delay and N is the filter order. The approximation error of a Lagrange interpolator in respect to an ideal fractional delay is lowest when the fractional delay D has been chosen so that $(N-1)/2 \le D \le (N+1)/2$. If the delay is in this optimal range, the filter is also *passive*, i.e., its magnitude response is less than or equal to one on the whole frequency range (Välimäki, 1995).

Recursive IIR filters may also be used for fractional delay approximation. A discrete-time filter, that has exactly flat magnitude response, has the transfer function of the form

$$A(z) = \frac{z^{-N} D(z^{-1})}{D(z)} = \frac{a_N + a_{N-1} z^{-1} + \ldots + a_1 z^{-(N-1)} + z^{-N}}{1 + a_1 z^{-1} + \ldots + a_{N-1} z^{-(N-1)} + a_N z^{-N}}, \tag{3.12}$$

where N is the filter order, $D(z)$ is the denominator polynomial, and a_k ($k = 1,2,\ldots,N$) are the real-valued filter coefficients. Filters of this type are called *allpass filters*.

The only fractional delay allpass filter design that has a closed-form solution is the maximally flat group delay design, also called the Thiran interpolator (Laakso *et al.*, 1996). The coefficients of the Thiran interpolator of order N are

$$a_k = (-1)^k \binom{N}{k} \prod_{n=0}^{N} \frac{D-N+n}{D-N+k+n}, \qquad k = 1,2,\ldots,N. \tag{3.13}$$

A close-to-minimum approximation error is obtained when the delay is

$$N - 0.5 \le D < N + 0.5, \tag{3.14}$$

a more exact range for the best possible accuracy for several Thiran allpass filters of different orders is given in (Välimäki, 1995).

For the first-order Thiran allpass filter with transfer function

$$A(z) = \frac{a_1 + z^{-1}}{1 + a_1 z^{-1}}, \tag{3.15}$$

the coefficient a_1 is given by

$$a_1 = \frac{1-D}{1+D}. \tag{3.16}$$

The range for the delay D can be chosen as $0.5 \le D < 1.5$ (cf. equation. 3.14). Tolonen (1998) suggests the usage of range $0 < D \le 1$ for practical application, but letting $D \to 0$ gives an asymptotically unstable filter, because the pole of the filter approaches the unit circle. Usage of such filter is not recommended (Välimäki, 1995). In the implementation of the guitar model for this thesis, real-valued delay line lengths are obtained using a first-order Thiran interpolator.

3.2 Extended String Model with Dual-Polarization Vibration

To take into account the effects that result from the differences in horizontal and vertical string vibration discussed in section 2.3.2, the string model may be extended with a second string loop. The excitation signal is divided to the two string loops after a multiplication by m_p and $1 - m_p$, where m_p is the input mixing coefficient. Respectively, the outputs of the string models are mixed together using the output mixing coefficient m_o. The parameters of the two models are slightly mistuned to achieve two-stage decay and beating effects. A string model with dual-polarization vibration is depicted in figure 3.7.

3.2.1 Sympathetic Couplings

Using the extended string model with both horizontal and vertical vibration polarizations, it is also feasible to implement the sympathetic couplings between strings. The sympathetic vibrations can be implemented simply by feeding the outputs from all the strings somehow filtered to the inputs of the string, but this approach is potentially un-

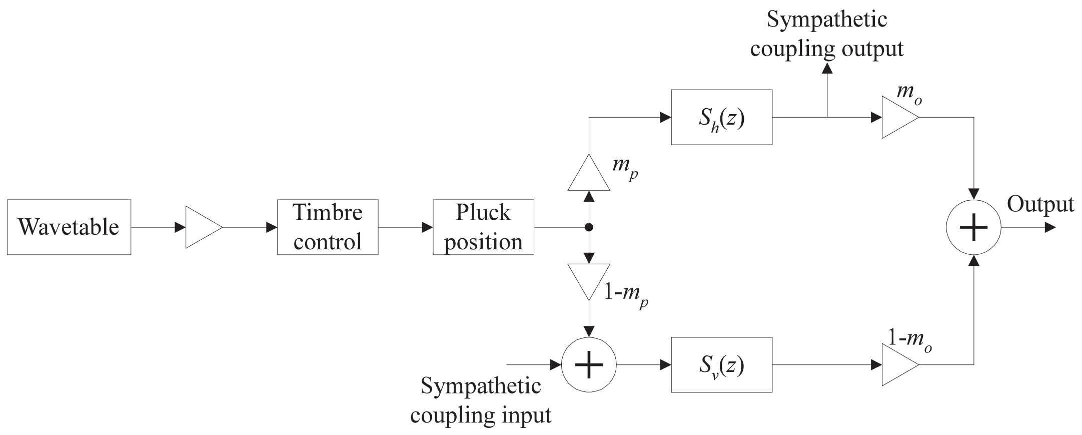

Figure 3.7: An extended string model with dual-polarization vibration and sympathetic coupling input and output, after (Karjalainen *et al.*, 1998)

stable. The coupling implementation described by Tolonen *et al. (*1998b) and portrayed in figure 3.7, is inherently stable. In this approach the excitation for sympathetic vibrations is taken from only one of the parallel strings modeling the two polarizations. To avoid feedback, the excitation is added to the input of those parallel strings that do not have sympathetic coupling output.

The amount of coupling is determined by a matrix **C** of coupling coefficients. This matrix can be written as

$$\mathbf{C} = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1N} \\ c_{21} & c_{22} & \dots & c_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ c_{N1} & c_{N2} & \dots & c_{NN} \end{bmatrix}, \tag{3.17}$$

where N is the number of advanced string models and the matrix elements c_{ij} $(i, j = 1, 2, \dots, N)$ are the gains of the *i*th horizontal string output to be sent to the *j*th vertical string input. Note that when $i = j$, the gain represents the coupling of the vibration polarizations of one advanced string model. While it is physically motivated to have real numbers less than one in absolute value as coefficients, the stability of the algorithm does not depend on these values.

3.3 Nonlinear Effects in Strings

The string model thus far presented is completely linear, and is realistic only if the string vibrations are sufficiently small in amplitude, and the string terminations are rigid. In plucked string instruments, there are, however, several nonlinear mechanisms, that remarkably affect tone quality. The propagation speed of a wave travelling the string is not constant, but varies with time. If either of the string terminations is not completely rigid, the nonlinear vibration provides also a mechanism for coupling of vibrational modes. Highly nonlinear effects result, if the amplitude of string vibration is strictly limited by, e.g., the frets or the fingerboard, as is the case in “slapping” playing techniques on the electric bass. It is fairly rare for the string to rattle against the frets in classical guitar playing, but flamenco guitars with lower action are more susceptible to this effect.

3.3.1 Tension Modulation

The most important reason for nonlinearity of a vibrating string is the modulation of the string length. Any displacement of the string results in an increase to its length, thus modulating the string tension and the propagation speed of the transversal vibration. Generally the speed of the longitudinal vibration is much larger than that of the transversal vibration, so that the variation of the tension may be considered to be immediately spread over the whole string.

The speed *c* of the transversal vibration can be written as

$$c = \sqrt{\frac{T_d}{\rho}}, \tag{3.18}$$

where T_d is the dynamic tension of the string containing the nominal tension and its deviation, and ρ is the linear mass density of the string. Because the relation between the fundamental frequency and the average propagation speed is linear, the fundamental frequency is expected to decrease gradually towards the nominal value as the amplitude of the string vibrations decrease over time. This is indeed the case, as may be seen from the fundamental frequency vs. time -graphs analyzed from recorded acoustic guitar tones, presented in figure 3.8. The fundamental frequency detection method is based on autocorrelation function with peak detection and parabolic interpolation, and has been formerly used, e.g., by Tolonen (1998).

In model-based guitar sound synthesis, the effect of tension modulation may be implemented using a time-varying fractional delay (TVFD) filter, in which the delay parameter is signal-dependent (Välimäki *et al.*, 1998). The general framework of a signal-

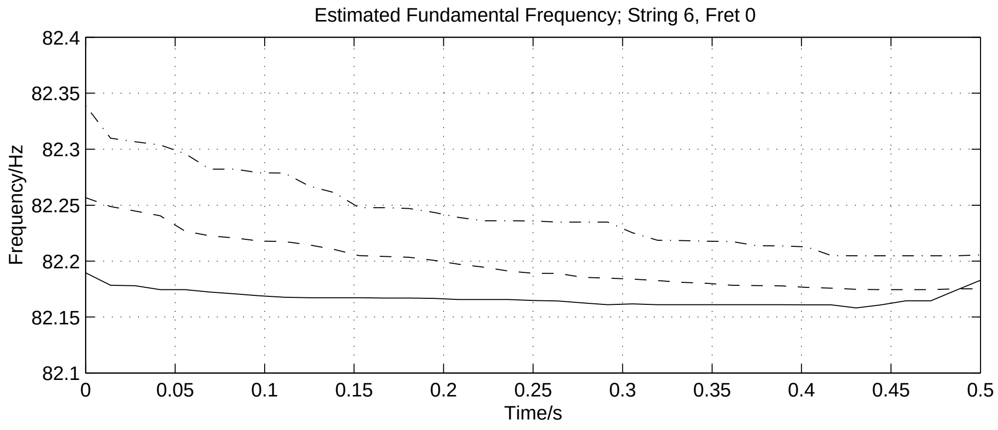

Figure 3.8: Examples of fundamental frequency evolution of guitar tones. Different dynamic levels are represented by differing line styles: solid for piano, dashed for forte and dashdot for fortissimo.

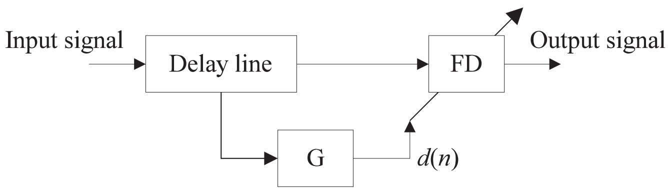

Figure 3.9: A general nonlinear delay line model, after (Välimäki *et al.*, 1998).

dependent delay line is depicted in figure 3.9. In the model the function *G* maps the delay line signal to a delay parameter $d(n)$, in which *n* is the discrete time index.

Välimäki *et al.* (1999) have shown that string tension modulation can be simulated by controlling the TVFD filter with a power-like signal. This approach applied to the single delay loop string model is shown in figure 3.10. The power estimation block in the figure consists of summing the first sample of the delay line with the last one, the second sample with the second last one, and so on. These sums are then squared and summed up to a single estimate of the delay line power. The filter $I(z)$ approximates temporal integration that is needed when the time-varying resampling is lumped to a single point at the end of the delay line (Tolonen *et al.*, 1999).

The greatest computational cost in the model comes from the power estimation. A simple way to reduce the cost is to approximate the power by a *sparse squared sum*, where only every *M*th sample pair is included in the calculation. Remarkable savings can be achieved without compromising the sound quality by using sparse squared sum with, e.g., $M = 6$ (Välimäki *et al.*, 1999).

3.3.2 Amplitude-Limiting Nonlinearities

An interesting nonlinear model has been described by Rank and Kubin (1997) for dual delay line simulation of the slapping technique on the electric bass. In this playing style, the string may be either struck using the knuckle of the thumb or pulled strongly outwards with right-hand index or middle finger. In both of these cases, the string hits the frets or the fingerboard during the first several fundamental periods of the tone. Striking a string with the knuckle is similar to the string-hammer interaction in piano, and the strongly pulled string may be modeled as having approximately triangular initial displacement.

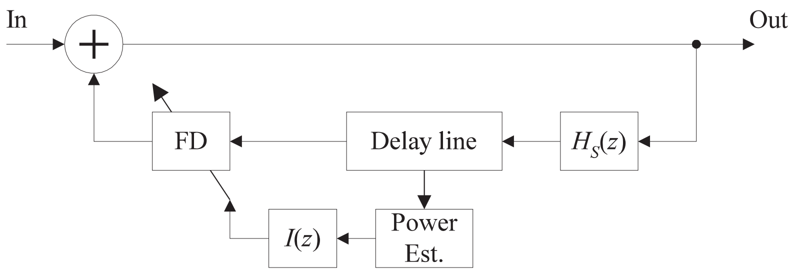

Figure 3.10: A single delay loop string model with tension modulation modeled using a TVFD filter, after (Välimäki *et al.*, 1999).

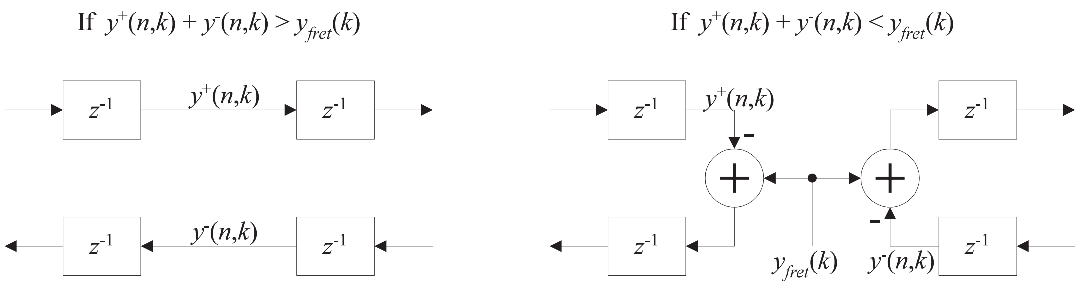
If $y^+(n,k) + y^-(n,k) > y_{fret}(k)$
If $y^+(n,k) + y^-(n,k) < y_{fret}(k)$
z^{-1}
$y^+(n,k)$
$y^-(n,k)$
$y_{fret}(k)$
-

Figure 3.11: Amplitude limitation modeling as displacement-conditional reflection. If the string displacement is below the fret level $y_{fret}(k)$, the waves on the delay lines are reflected and the fret distance is added to both of the delay lines. The reflection reverses the phase. After (Rank and Kubin, 1997)

The contact between the string and the frets results in a nonlinear limitation of the string vibration amplitude. To facilitate this limitation requires testing the string displacement against the limits set by the limiting fret. This test can be implemented in a straightforward manner if displacement is selected as the wave variable. The actual limitation is implemented by reflecting the waves on the two delay lines on the contact position. The reflection is phase-inversive and the distance of the fret is added into both of the delay lines, so that the amplitude is limited to the proper value (see figure 3.11). Note from the way the condition has been formulated, that the limiting is assumed to lie below the string.

To produce strong notes on an acoustic guitar requires a sufficient amount of string vibration perpendicular to the soundboard. This leads to an apparent increase of the fret-rattle risk. It is nevertheless possible to produce sounds with both body and volume because the vibration pattern of the string is anti-symmetrical. In figure 3.12 the two extremes of an ideal string vibration after release are depicted. The figure suggests that if the string is pulled away from the soundboard near the bridge, it will easily rattle against the frets. This technique is called *Bartok pizzicato* and it is sometimes used in contemporary guitar music. However, if the string is pushed towards the soundboard, as in normal apoyando and tirando strokes, the risk of fret-rattle is considerably reduced.

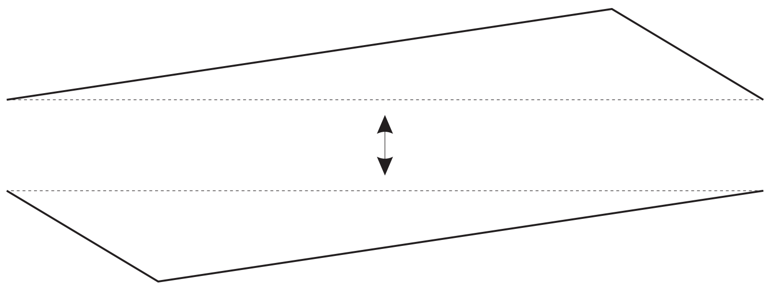

Figure 3.12: Two extremes of an ideal string vibration.

3.4 Modeling the Guitar Body

In the beginning of this chapter, the guitar model was described to consist of three functional substructures: the excitation, the vibration of the strings and the radiation from the instrument body. It is convenient to preserve this partition when developing a computational model of an acoustic guitar. In the previous sections, an effective and detailed model of the string vibrations was presented. To obtain high-quality synthetic guitar sound, the response of the guitar body must be introduced into the instrument model.

3.4.1 Commuted Model of Excitation and Body

Direct modeling of the guitar body response by digital filters would require filters of impractically high order for real-time applications, because of the complexity of the vibrating structure (Karjalainen and Smith, 1996). Commuted waveguide synthesis (CWS) approach can be utilized to include the body response in the guitar model in an efficient manner (Smith, 1993; Karjalainen *et al.*, 1993). This approach is based on the theory of linear systems.

In CWS the parts of the instrument are represented as linear filters $E(z)$, $S(z)$ and $\mathrm{B}(z)$ for excitation (plucking), string vibration and body radiation, respectively. The filter system is excited with an impulse $\delta(n)$. The output signal is now the convolution of the impulse responses $e(n)$, $s(n)$, and $b(n)$ of the three filters $E(z)$, $S(z)$, and $B(z)$, and the unit impulse $\delta(n)$,

$$y(n) = \delta(n) * e(n) * s(n) * b(n) = e(n) * s(n) * b(n), \tag{3.19}$$

where $*$ denotes the convolution operator.

In the Laplace- or z-transform domain the equation 3.19 can be expressed as a product

$$Y(z) = E(z)S(z)B(z). \tag{3.20}$$

Because the instrument parts are represented by linear components, the principle of commutation can be applied, and the equation 3.20 can be written as

$$Y(z) = B(z)E(z)S(z). \tag{3.21}$$

It is advantageous to precompute the impulse responses $b(n)$ and $e(n)$ into a single impulse response $x_{exc}(n)$ that is then stored in a wavetable and used to excite the string model as seen in figure 3.13.

3.4.2 Body Resonators

In commuted synthesis the computational complexity is reduced, but memory requirements are increased. The excitation signal contains components from the attack portion

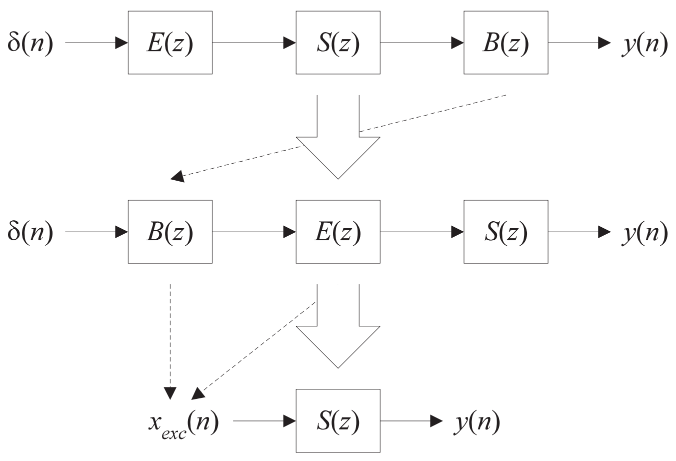

Figure 3.13: Commuted waveguide synthesis. The instrument model is represented by three linear filters on the top. In the middle, the order of the filters is changed. On the bottom, the body and excitation models are precomputed into a single excitation signal $x_{exc}(n)$. After (Tolonen, 1998).

of the guitar sound as well as a significant contribution from the few lowest body resonances. The attack part is fairly short, and the length of the needed excitation signal is determined by the most prominent body resonances (Tolonen, 1998). If these resonances are implemented separately, the excitation signal can be truncated, and the instrument sound can be adjusted by altering the parameters of the body resonators.

Separate body resonators can be implemented either in cascade or in parallel with the string models. In cascade implementation the string model is fed with an excitation signal $x_{exc}(n)$, where the body resonances are absent. After the string model, the resulting signal is fed to the body resonator filters. In parallel implementation, the body resonators are fed with separate excitation signals $x_{R1}(n)$ and $x_{R2}(n)$, and the output of the filters is summed to the output of the string model. A hybrid of the two alternatives can be implemented by cascading the two resonators, while still keeping them parallel to the string model. Along the lines of (Tolonen, 1998), where comparison of the alternatives as well as procedures for computation of the needed excitation signals are described, parallel body resonators were chosen for this work.

In this work second-order IIR filters with transfer function

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}, \tag{3.22}$$

are used for each of the separately implemented body resonators. For a second-order peak filter suitable for parallel resonator implementation, the coefficients can be given as (Orfanidis, 1996)

$$\begin{cases} b_0 = (1-\beta) \\ b_1 = 0 \\ b_2 = (\beta - 1) \\ a_1 = -2\beta\cos(\omega_0) \\ a_2 = (2\beta - 1) \end{cases}, \tag{3.23}$$

where

$$\omega_0 = 2\pi f_{n0} \tag{3.24}$$

and

$$\beta = \frac{1}{1 + \tan(\pi\,\Delta f_n)}. \tag{3.25}$$

The parameters f_{n0} and Δf_n are the center frequency and the 3dB bandwidth of the resonator, respectively. Both are given on normalized frequency scale, 0.5 corresponding to half of the sampling rate, or the Nyquist frequency. An example of a resonance filter response with center frequency corresponding to 203Hz and bandwidth corresponding to 10Hz with a sampling rate of 2205Hz, can be seen in figure 3.14.

3.5 Model Excitation

For expressive guitar synthesis, the implementation of different properties of the plucking process is crucial. Different plucking techniques, the variation of pluck position along the string as well as the more subtle changes in plucking process must somehow be modeled, in order to achieve natural-sounding guitar synthesis. The following subsections discuss the implementation of these properties in synthesis.

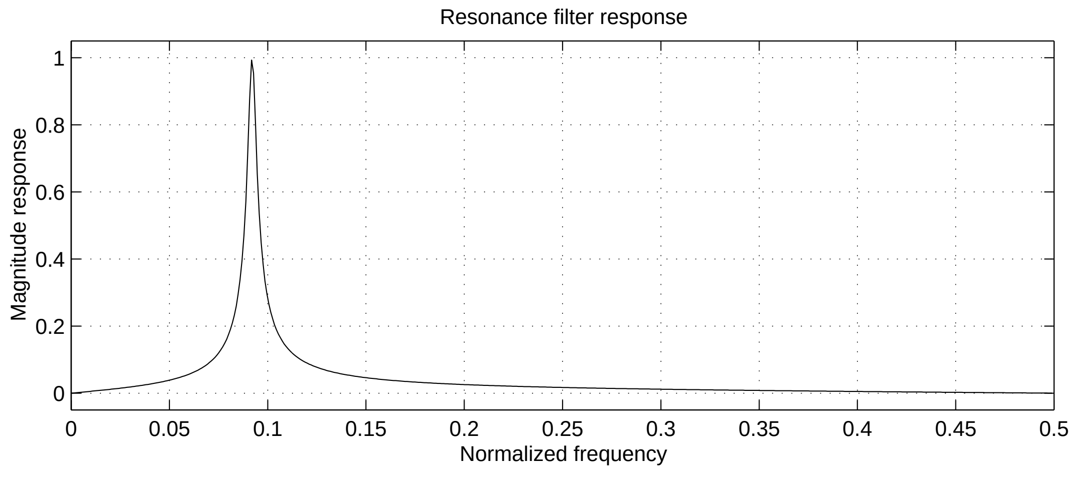

Figure 3.14: Example of a resonance filter response.

3.5.1 Pluck Position

As was seen in section 3.1.3, the effect of the plucking position introduced in section 2.2 can be realized in the digital domain using a comb filter structure illustrated in figure 3.15. In the comb filter the delay z^{-l} corresponds to the actual distance from the plucking position to the guitar bridge, and the coefficient c, where $c \geq 0$, is proportional to the losses in the string from the plucking position to the bridge.

The delay of the comb filter corresponds to the time it takes for the excitation to travel the left-side route around from $E2$ to $E1$ in figure 3.4. In practice, however, the normally shorter right-side route from point $E1$ to $E2$, may be used.

If the total delay along the string loop is L samples and the delay corresponding to the route from point $E1$ to $E2$ is l samples, then the delay of the left-side route from $E2$ to $E1$ is $L - l$ samples. Now the transfer function of the comb filter for the right-side route is

$$H_R(z) = \frac{1}{2}(1 - cz^{-l}), \tag{3.26}$$

and the corresponding filter for the left-side route is

$$H_L(z) = \frac{1}{2}[1 - cz^{-(L-l)}]. \tag{3.27}$$

The squared magnitude response of the filter H_R can be written as

$$\left|H_R(e^{j\omega})\right|^2 = H_R(e^{j\omega})H_R(e^{-j\omega}) = \frac{1}{4}(1 - 2c\cos(\omega l) + c^2), \tag{3.28}$$

similarly

$$\left|H_L(e^{j\omega})\right|^2 = \frac{1}{4}(1 - 2c\cos(\omega L - \omega l) + c^2). \tag{3.29}$$

Setting the two magnitude responses equal yields

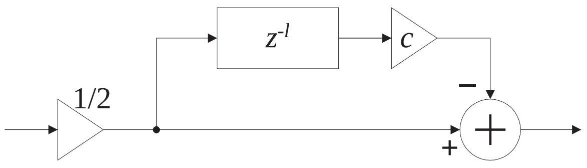

Figure 3.15: The comb filter structure used to implement the effect of plucking position

$$\begin{aligned} &\cos(\omega L - \omega l) = \cos(\omega l) \\ &\Rightarrow \omega(L - l) = \pm\omega l + m2\pi, \qquad m = 0, 1, 2, \ldots \\ &\Rightarrow \omega = \frac{m2\pi}{L - 2l} \quad \vee \quad \omega = \frac{m2\pi}{L} \end{aligned} \tag{3.30}$$

The latter of the resulting relations gives exactly the radian frequencies of the fundamental and the harmonics of the string of length *L*. Thus, on the most important frequencies the filters corresponding to the right-side route and the left-side route have equal magnitude responses.

3.5.2 Plucking Style

In commuted guitar synthesis, the properties of the plucking process are contained in the excitation signal, as described in section 3.4.1. The excitation signal must thus be different for different types of excitation. In addition, the excitation signal depends on string and fret position. Also the excitation signal for the parallel body resonators are calculated from samples of natural guitar sound, and may be different for each sound. The direct approach of using the calculated excitation signals leaves us with a great number of quite similar signals, and requires a great amount of memory capacity. The number of signals must thus be reduced somehow.

The most straightforward way of cutting down the number of necessary signals is to use the same excitation signals for a small group of tonally similar sounds. To be able to still produce high-quality synthesis, the similarity of the signals must be verified either by listening or by some objective means. This task has not been addressed in the literature, and further research is needed.

The differences in the string excitation signals may be partly compensated, and additional expressive control achieved by using a timbre control filter for the excitation signal (see figure 3.6). It can also be used to provide slight random variations to the sound, to give a less monotone synthesis result.

The pizzicato effect described in section 2.3.5 could be implemented by adjusting both the properties of the string model as well as the excitation signal. The frequency dependent damping caused by the right-hand palm may be realized by increasing the overall attenuation of the string model loop filters and adjusting the filter pole so that higher frequencies are suppressed. The timbre control filter can be used to adjust the properties of the excitation signal to get the desired effect.

3.6 Multirate Model Structures

The high-frequency components of guitar sound are quickly damped after the attack part of the sound. This suggests that multirate implementations, where the decay part of the sound is synthesized at a lower sampling rate than the attack part, can yield efficient synthesis models (Smith, 1993). If the two lowest body resonances are implemented separately as described in section 3.4.2, the resonators can also be operated at a much lower sampling rate than the string models (Välimäki and Tolonen, 1997, 1998). The following discussion assumes basic knowledge on sampling rate conversion. For further information see, e.g., Proakis and Manolakis (1992).

In the model suggested by Tolonen (1998), the shared body resonators are implemented using a sampling rate of 2205 Hz, the sum of the resonators is then upsampled to 11025 Hz and added to the sum of the output signals of the string models, which operate at 11025 Hz. Then this sum is upsampled by the factor of 2 and the attack parts of the tones are produced at rate 22050 Hz, and added to the output signal.

The purpose of the lower sampling rates is to yield computational savings over a single-rate model. Thus the structures of the sample rate converters must be sufficiently effective not to ruin the possibility of improved performance. Given the special conditions of the guitar model, it is possible to use simpler approaches than in the general case.

Referring to the figure 3.14, it can be noted that even with a sampling rate of 2205 Hz the body resonator peaks are close to zero frequency and hence it suffices to suppress aliased signal components near the multiples of the original sampling frequency. This can be achieved by using a filter with zeros on angular frequencies

$$\omega = \frac{2\pi n}{M}, \qquad n = 1,\ldots,M\text{-}1, \tag{3.31}$$

where M is the upsampling factor. The impulse response of this kind of filter consists of M taps of values $1/M$, and they are called *recursive running sum* (RRS) filters (Rabiner and Gold, 1975).

For the upsampling of the string model output signals by the factor of 2, a more complex filter structure is needed. For this, a sparse linear-phase FIR filter may be selected. For a 4Nth order halfband filter only N+1 filter coefficients need to be stored in computer memory, since every other coefficient equals zero and the filter is symmetric (Tolonen, 1998). As figure 3.16 shows, aliased components may be suppressed by more than 50 dB using a filter of order 60 with 16 distinct coefficient stored in computer memory.

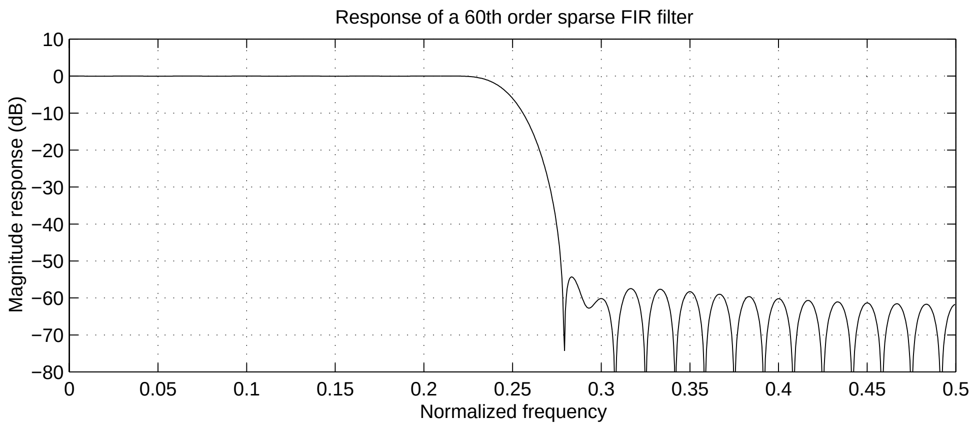

Figure 3.16: Magnitude response of a sparse halfband FIR filter of order 60.

4 Guitar Model Implementation

In this chapter, the real-time implementation of the guitar model is described. The implementation uses object oriented programming paradigm (Rumbaugh *et al.*, 1991) and C++ programming language (Lippman, 1991). The program development has been done on Silicon Graphics IRIX platform, but general portability has been an important design issue.

The implemented components have been divided to two libraries; general purpose digital signal processing components are included in the C++ class library called LibRdsp while the LibRdim library contains the higher-level instrument model implementations. The signal processing library provides many components not presented in this thesis, and also tools for multi-channel signal processing, but this chapter concentrates solely on the one-channel processing units with relevance to the implemented guitar synthesizer.

The following sections present the abstract base classes (section 4.1), the implemented string models (4.2) and the guitar body model implementations (4.3) of the signal processing library. The instrument model base class and the class interface are described in section 4.4 and the complete guitar models in section 4.5. Section 4.6 discusses the issues relating to signal processing in workstation environment.

4.1 DSP-Library Classes

All the signal processing units in the LibRdsp library inherit the properties of a few abstract base classes. These classes include signal sources, drains, processors and filters. The inheritance of the base classes and the methods they provide are depicted in figure 4.1 using OMT notation (Rumbaugh *et al.*, 1991), and described in the following subsections.

4.1.1 Signal Source

`CSignalSource` is an abstract base class for all units that produce an output signal either from scratch or as a result of possibly quite complex filtering operations. All classes inherited from `CSignalSource` must implement the `Get` method, which returns a sample from the source.

4.1.2 Signal Drain

The natural counterpart for `CSignalSource` class is the `CSignalDrain` class. This abstract class is the base class for all units that need to have an input signal for

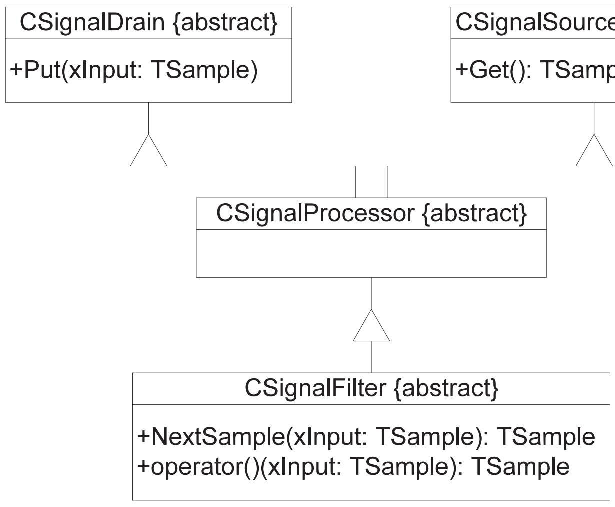

Figure 4.1: Base classes of the LibRdsp library

some purpose. Currently, there are no implementations of "pure" signal drains in the library, i.e. there are no classes derived only from the `CSignalDrain` class. Such drains could, however, include platform audio outputs and writing of sound files of different formats. There are plans to include these functionalities directly to the DSP library.

The operational part of all classes inherited from `CSignalDrain` is the `Put` method, which feeds the signal drain with one sample or sample frame.

4.1.3 Signal Processor

The `CSignalProcessor` abstract base class inherits its properties from both `CSignalSource` and `CSignalDrain`, i.e. classes inherited from `CSignalProcessor` can produce as well as consume a signal. Because the input and output signals need not have same sample rates, also sample rate converters and asynchronous signal processing blocks can be derived (but have not yet been derived) from this base class.

4.1.4 Signal Filter

`CSignalFilter` is a base class for all "normal" one-channel signal processing units that take one input sample for every output sample. This class implements the `NextSample` method so that it first calls the `Put` method and then the `Get` method of the derived class. The derived class must implement `Put` and `Get` methods so that when called in the aforementioned order, they produce the desired filtering effect. `CSignalFilter` also implements the parenthesis operator as an abbreviation for writing `NextSample()`.

The use of `NextSample` method instead of the `Put` and `Get` methods, may, in some cases, lead to reduced performance. This kind of situation may occur, if the actual non-abstract class of the signal processing element is known to the compiler, i.e. no base class pointers or function arguments are used. Then the use of base class method may result in one additional layer of pointer operations. In such cases it is advisable to use the `Put` and `Get` methods of the derived class instead.

4.1.5 Classical Filter Structures

For classical DSP filter structures that can conveniently be described as numerator and denominator coefficients b_i and a_i of a filter transfer function of form

$$H(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_M z^{-M}}{1 + a_1 z^{-1} + \ldots + a_N z^{-N}}, \tag{4.1}$$

LibRdsp has the `CFilter` base class (see figure 4.2). `CFilter` is derived from `CSignalFilter` class, i.e. it has a method called `NextSample`, which feeds the next input sample to the filter and returns the corresponding output sample from the filter.

`CFilter` class provides a `Get` method, which simply returns the value of a protected variable called `M_xCurrentSample`, i.e. filter implementations only need to provide a `Put` method that leaves the filtering result in this variable. Filters also have methods for setting the filter coefficients, for resetting the filter memory, and for querying the order of the filter.

Filters derived from `CFilter` all have two different overloaded `Update` methods for setting the filter coefficients. The one with prototype "`int CFilter::Update(CFilterCoeffs &xrCoeffs)`" creates an internal copy of the filter coefficients in `xrCoeffs` so that if the state of `xrCoeffs` is later changed, it has no effect on the actual parameters used by this `CFilter` instance. On

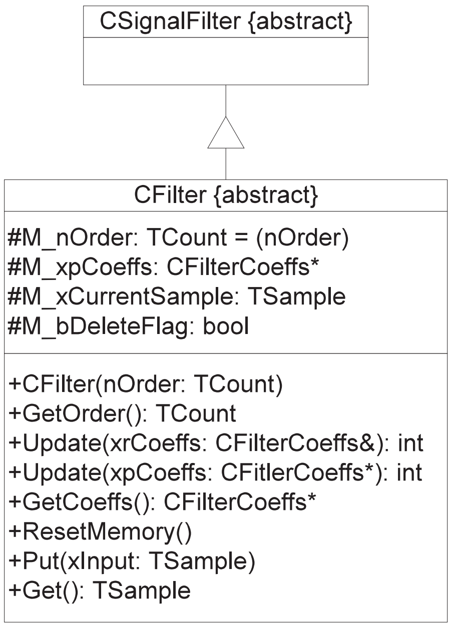
CSignalFilter {abstract}
CFilter {abstract}
#M_nOrder: TCount = (nOrder)
#M_xpCoeffs: CFilterCoeffs*
#M_xCurrentSample: TSample
#M_bDeleteFlag: bool
+CFilter(nOrder: TCount)
+GetOrder(): TCount
+Update(xrCoeffs: CFilterCoeffs&): int
+Update(xpCoeffs: CFitlerCoeffs*): int
+GetCoeffs(): CFilterCoeffs*
+ResetMemory()
+Put(xInput: TSample)
+Get(): TSample

Figure 4.2: CFilter base class

the other hand "`int CFilter::Update(CFilterCoeffs *xpCoeffs)`" causes the filter to use the coefficients pointed to by `xpCoeffs` pointer. Using this method it is quite easy to create a group of filters all using the same set of coefficients. This way, all the changes in the coefficients can be done only once. Care must however be taken when deleting the filter coefficients; the user must be sure that no filter is using the coefficients which are to be deleted.

Filter coefficients can also be accessed with `GetCoeffs` method that returns a pointer to the filters internal coefficients. If the coefficient instance that a filter is using was originally allocated by the filter during its initialization, the filter coefficients will also be deleted by the filter class destructor.

For filter coefficients LibRdsp provides the `CFilterCoeffs` class, which has storage space for `nOrder+1` numerator coefficients and for `nOrder` denominator coefficients, where `nOrder` is this the order of the filter coefficient structure (max$\{n,m\}$, see equation 4.1) The order is also a constructor argument for `CFilterCoeffs` class. `CFilterCoeffs` class has the methods `GetNumerator`, `GetDenominator`, `SetNumerator` and `SetDenominator` for accessing the coefficient values. Both the numerator and the denominator coefficients are indexed from 0 to filter order, but querying the denominator coefficient with index 0 always returns 1.0 and setting the same coefficient has no effect whatsoever.

4.1.6 Ring Buffer

`CRingBuffer` class implements a ring buffer to be used as an integer-length delay line. The class is also used as a memory block in the *N*th order filters of the signal processing library. The implementation is tested for fast performance and it does not check any of the parameter or signal values. The class has access methods for the ring buffer length, and for signal values along the buffer (see figure 4.3). The constructor argument is the desired maximum length of the buffer.

For efficient operation, the pointer update and indexing operations of the ring buffer are implemented using fast bit-manipulation operations. This way, no time-consuming integer divisions are needed. The indexing is based on a memory buffer with length of power of two. After an index has been increased or decreased, a bitwise 'and' operation is performed with the index and a mask integer. The mask integer has the value $2^n - 1$, where 2^n is the length of the memory buffer. Thus, all index values are restored to the proper range. For arbitrary integer ring buffer lengths, two indices are used, one for the head and one for the tail of the ring buffer.

4.2 String Model Implementations

For string models presented in chapter 3, LibRdsp has a few different class implementations. The simpler ones of these models are described in section 4.2.2 and the more advanced, dual-polarization model in section 4.2.3. All the implementations use common implementations of the necessary filters. These filter classes are described first.

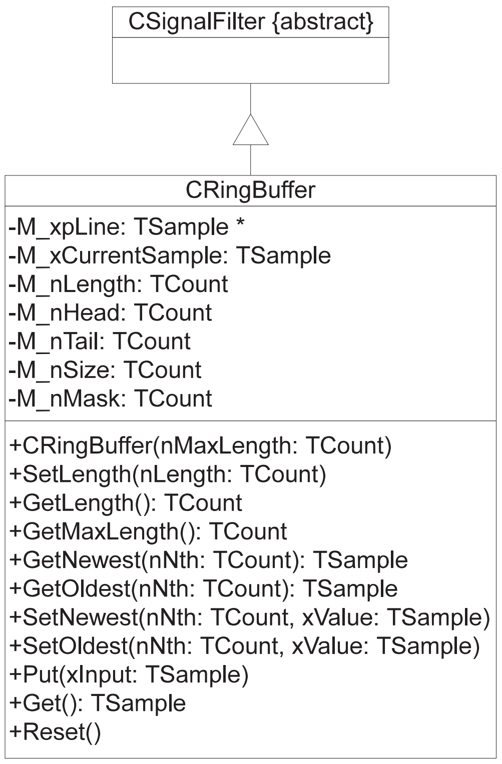
CSignalFilter {abstract}
CRingBuffer
-M_xpLine: TSample *
-M_xCurrentSample: TSample
-M_nLength: TCount
-M_nHead: TCount
-M_nTail: TCount
-M_nSize: TCount
-M_nMask: TCount
+CRingBuffer(nMaxLength: TCount)
+SetLength(nLength: TCount)
+GetLength(): TCount
+GetMaxLength(): TCount
+GetNewest(nNth: TCount): TSample
+GetOldest(nNth: TCount): TSample
+SetNewest(nNth: TCount, xValue: TSample)
+SetOldest(nNth: TCount, xValue: TSample)
+Put(xInput: TSample)
+Get(): TSample
+Reset()

Figure 4.3: Ring buffer class implementation.

4.2.1 Filter Classes

The class `CFilterAPL1` is a one-channel first-order allpole filter implementation, i.e., it implements the difference equation

$$y(n) = b_0 x(n) - a_1 y(n-1),. \tag{4.2}$$

where $b_0 = g(a_1 + 1)$ (cf. Equation 3.6). In addition to the two `Update` methods described in section 4.1.5, the allpole filter has a similar method with two double-precision floating-point arguments b_0 and a_1. This filter is a popular choice to be used as a loop filter in the SDL string models (Välimäki *et al.*, 1996; Tolonen, 1998), and can also be used as an excitation timbre control filter, as seen in figure 3.6.

`CCombFilter` class implements the pluck position filter of section 3.5.1. The class is inherited from `CSignalFilter` base class and has methods for setting and querying the parameters of the model: the delay line length L and the coefficient c (see figure 3.15).

To be able to adjust the length of the string model delay lines, fractional delay filters must be used, as discussed in section 3.1.4. The LibRdsp library implements both the Lagrangian and the Thiran interpolator. Currently, the Thiran interpolator is used in the string models, because of the exactly flat magnitude response. The `CFilterAP1` class implements the first-order allpass filter. In addition to the two normal `Update` methods, the class an `UpdateDelay` method, which takes the desired fractional delay as an argument, and updates the filter coefficient accordingly.

4.2.2 Single Delay Loop String Models

The class `CSimpleString` implements a simple single delay loop string model. The structure of the string model is depicted in figure 4.4. The model consists of the delay line z^{-L} (`CCombFilter`), the fractional delay filter $F(z)$ (`CFilterAP1`), and the loop filter $H_l(z)$ (`CFilterAPL1`). The class has separate methods for accessing each of the model's parameters.

The class `CSDLString` augments the basic string model by adding a first-order all-pole filter in front of the actual string loop. With this additional filter it is possible to control the tone quality of the string model to some extent. Although a simple model, this class can yield relatively high-quality synthesis already with some expressional capabilities.

4.2.3 Dual-Polarization Model

The class `CTwoPolarString` implements the string model with two vibration polarizations, as described in section 3.2 (see also figure 3.7). Figure 4.5 presents the implementational structure of the dual-polarization string model class. The timbre control unit of the model is a first order allpole filter of type `CFilterAPL1`, the pluck position filtering is done with a `CCombFilter` instance, and the two vibration polarizations are implemented as `CSimpleString` string models.

The greatest desired length of the string model in samples must be given as a constructor argument for memory allocation. All other parameters of the string model are initialized to default values during instantiation, and can be changed using the `SetParam` method. Parameter values may be queried with the corresponding `GetParam` method. A shared method for all parameter updates was chosen because of the number of parameters would make the number of methods impractically large, if the updates were handled by separate methods. The parameter update and query methods use an enumerated list of possible arguments (type `ETwoPolarStringParam`).

For sympathetic coupling implementation, the `CTwoPolarString` class has two methods: `CouplingPut` and `CouplingGet` for feeding and getting, respectively, one sample from/to the string model.

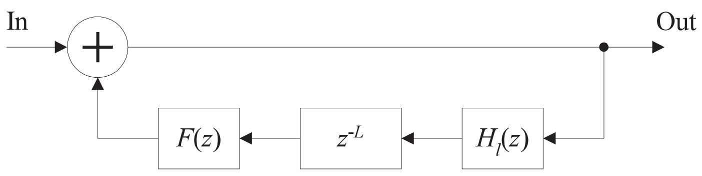

Figure 4.4: `CSimpleString` block diagram.

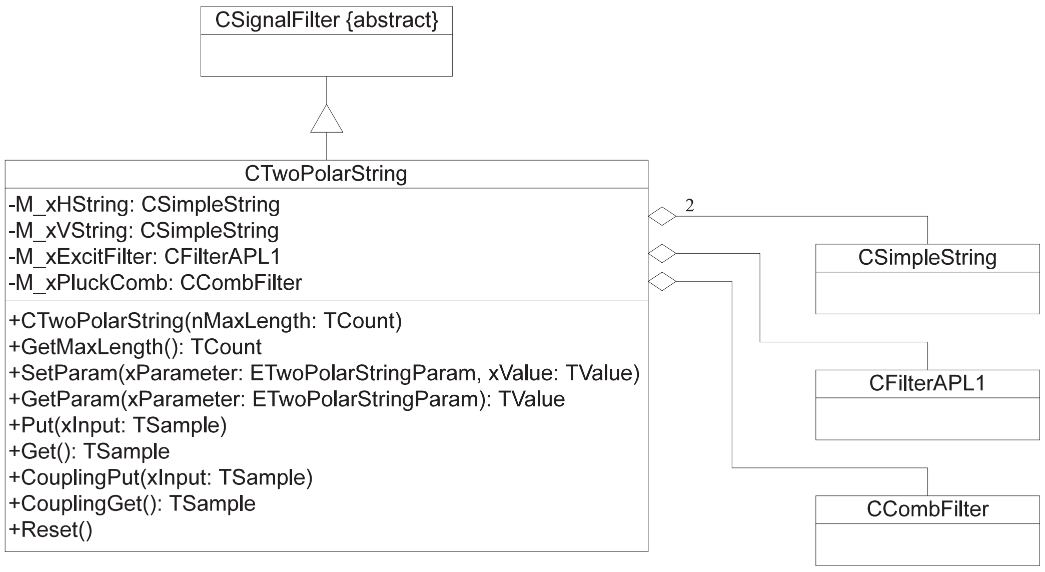
CSignalFilter {abstract}
CTwoPolarString
-M_xHString: CSimpleString
-M_xVString: CSimpleString
-M_xExcitFilter: CFilterAPL1
-M_xPluckComb: CCombFilter
+CTwoPolarString(nMaxLength: TCount)
+GetMaxLength(): TCount
+SetParam(xParameter: ETwoPolarStringParam, xValue: TValue)
+GetParam(xParameter: ETwoPolarStringParam): TValue
+Put(xInput: TSample)
+Get(): TSample
+CouplingPut(xInput: TSample)
+CouplingGet(): TSample
+Reset()
2
CSimpleString
CFilterAPL1
CCombFilter

Figure 4.5: The dual-polarization string model class composition.

4.3 Body Model Implementations

The most straightforward way to incorporate the body response in the guitar model sound is the commuted synthesis method presented in section 3.4.1. This approach requires only implementation of wavetables in addition to the actual string models. The more versatile body model implementation with shared resonators is a little more demanding.

4.3.1 Excitation Wavetables

Wavetable is a memory structure that stores a prerecorded or -computed signal segment for retrieval at a later time. In DSP applications, the signal may be fetched one sample at a time, so that the wavetable signal source class must store a pointer to the current sample of the segment and return the next sample as desired. For some applications, it would be good to have multiple such pointers, to be able to get many samples at a time from different locations of the wavetable.

The class `CVectorSource` is a wavetable signal source class, derived from the `CSignalSource` abstract base class. It has methods for loading the signal from file, setting and getting individual sample values, getting the signal length, and getting the next sample from the memory buffer. For additional pointers to the wavetable, user can use the `CVectorSourceIter` class that takes a pre-initialized `CVectorSource` instance as a constructor argument, and provides a new output for the same signal.

4.3.2 Shared Resonators

Shared body resonators from section 3.4.2 can be implemented as standard second-order IIR filters. For second-order, direct form I filters (Jackson, 1989), the LibRdsp library has the class `CFilterDFI2`. Note that a special implementation for the body resonator type filter would be advantageous because one of the five filter coefficients in equation 3.23 is zero. Taking this into account, however, will not lessen the computational burden significantly, especially, when the resonators function at a low sampling rate compared to the complex string models.

For calculation of the resonator filter coefficients, the library has an auxiliary function `SecondOrderPeakFilter`, which takes the center frequency and the bandwidth of the desired resonance on a normalized frequency scale as input arguments, and calculates the filter coefficients into a preallocated or into a new `CFilterCoeffs` instance.

4.4 Instrument Model Base Class

All instrument models in the implemented instrument library LibRdim are derived from an abstract base class `CInstrModel` (figure 4.6) that in turn inherits the properties of the `CSignalSource` class of section 4.1.1. The `CInstrModel` interface contains the methods for controlling the parameters of the instrument models, for setting and querying the sample rate, and for getting the next sound sample from the instrument model. The parameters of the model are controlled by the `React` method, which takes one ASCII control message as an argument, does the necessary operations on the parameters, and returns an acknowledgement message.

4.5 The Aggregate Guitar Model

Guitar models that inherit the properties of the `CInstrModel` base class combine all the elements described in earlier sections of this chapter. Excitation signals from

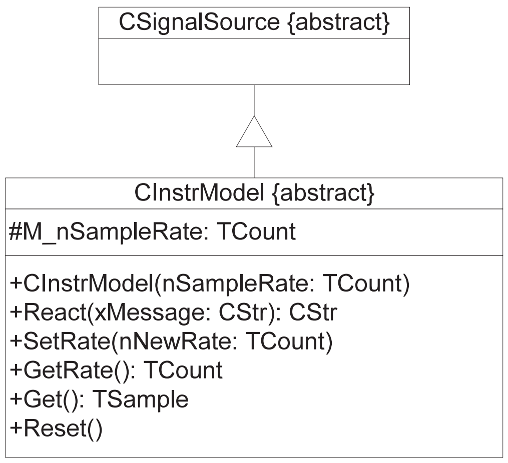

Figure 4.6: Instrument model abstract base class.

wavetables are filtered according to the plucking position and the desired timbre. String models are fed with the equalized signal, and body model outputs are added to the string outputs to give the end result.

4.5.1 Model Structure

The functional structure of the implemented guitar model is depicted in figure 4.7. The model has *N* dual polarization string models, and two lowest body resonances are implemented in parallel to the strings.

The strings are of the type `CTwoPolarString` and they run at a sampling rate of 22050 Hz, which is also the output sampling rate of the whole model. Each string may have multiple different excitation signals of type `CSignalVector`, e.g., one for each fret along the string. The signals needed for sympathetic coupling between the strings are taken from the special output, multiplied by the matrix **C** of coupling coefficients, and fed back to the string models' coupling signal input.

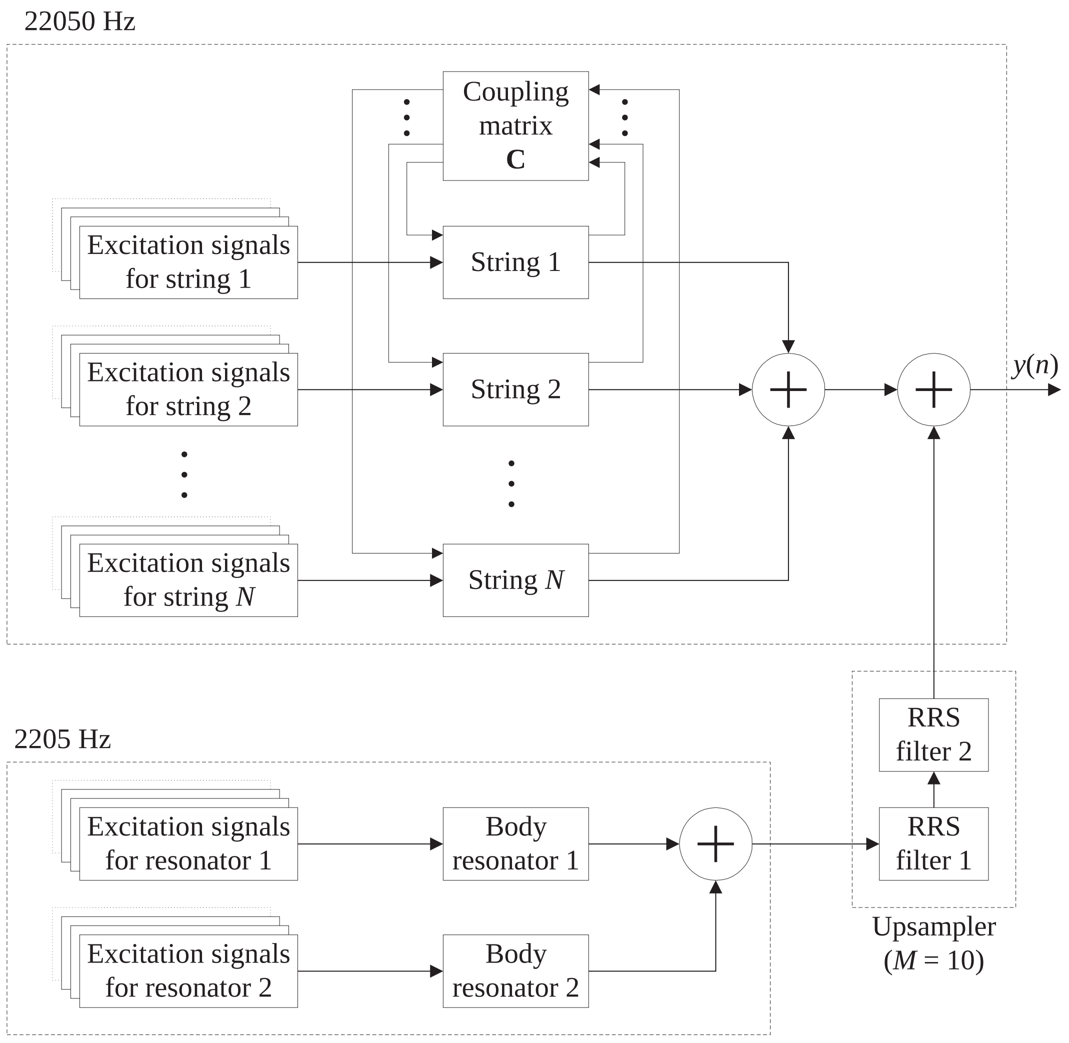
22050 Hz
Coupling matrix C
Excitation signals for string 1
String 1
Excitation signals for string 2
String 2
Excitation signals for string N
String N
y(n)
RRS filter 2
RRS filter 1
Upsampler (M = 10)
2205 Hz
Excitation signals for resonator 1
Body resonator 1
Excitation signals for resonator 2
Body resonator 2

Figure 4.7: Block diagram of the implemented guitar model with sympathetic coupling and shared body resonators running at a lower sampling rate than the strings.

The two shared body resonators are of type `CFilterDFI2`, as described in section 4.3.2. The resonators operate at a sampling rate of 2205 Hz, i.e., resonator output signals must be upsampled by factor $M = 10$ before adding to the string output signal. The conversion is done with two cascaded recursive running sum filters. It is noted that the first of the filters can be readily implemented by observing the output values of the resonators ten times before calculating the next output values. The second filter can be efficiently implemented using the structure depicted in figure 4.8. This recursive structure for the finite impulse response filter is based on the identity

$$H_{RRS}(z) = \frac{1}{M}\sum_{k=0}^{M} z^{-k} = \frac{1-z^{-M}}{M(1-z^{-1})}. \tag{4.3}$$

4.5.2 Guitar Model Classes

The guitar model can easily be implemented so that it permits having any reasonable number of strings. This kind of a structure would allow easy adaptation to the synthesis of instruments with different number of strings. However, the inclusion of the sympathetic vibrations makes such a general model quite unfeasible, because the need for two nested for-loops for the matrix multiplication possibly makes the implementation very inefficient. If the number of strings in the model is constant, the for-loops may be "unrolled" to give better performance on modern pipelined processors.

In LibRdim library the six-string guitar model with dual-polarization strings and separate body resonators is called `CAdvGuitar6Str`. The model is initialized by giving the desired sampling frequency and the lowest possible fundamental frequencies of the strings. Other methods are as described for the abstract instrument model class in section 4.4. The `React` method controls the parameters of the model using a new control protocol that is described for the guitar model in section 5.5.

4.6 Signal Processing on Workstation Platform

As the computational capacity has grown, interactive sound synthesis and processing applications on PC and workstation platforms have gained increasing interest. General purpose microprocessors have been developed to take into account the requirements posed by real-time audio and multimedia. Operating systems have also been developed

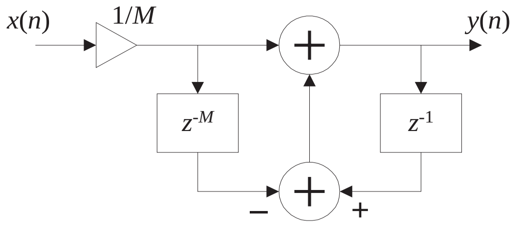

Figure 4.8: Recursive running sum filter structure.

along these lines, however, few vendors specify and much less guarantee the temporal, real-time performance of their operating system.

4.6.1 Signal Processors vs. Microprocessors

RISC (Reduced Instruction Set Computer) and CISC (Complex Instruction Set Computer) microprocessors and specialized signal processors have very different performance profiles. As general purpose microprocessors need to be effective in many different computational tasks, their structure is more compromised than the structure of the specialized signal processors.

One thing that has traditionally differenced the signal processors from other microprocessor families is the existence of the Multiply-Accumulate (MAC) instruction for efficient implementation of the inner product calculation often needed in signal processing applications. Signal processors typically have also multiple memory busses, so that the operands of the calculations may be fetched in a parallel fashion, and the performance of the loop constructions is very fast.

The structure and instruction set of modern microprocessors have been influenced by the paradigms used in DSPs. Processors often implement the MAC instruction and it has been estimated that a RISC processor with the MAC instruction is capable of performance competition with signal processors (Weiss, 1996).

4.6.2 Operating Systems and Sound Subsystems

As the processors currently have enough capacity to do reactive sound synthesis and processing, the next critical component is the operating system. Real-time multimedia requires extensions or redesign of the operating systems. In current multiprocessing operating systems, the system scheduler gives time slices of processor time to the processes one at a time. The scheduler seldom gives any guarantee of the temporal performance of the system, and give only limited means of controlling the priority of the processes to the user.

In reactive audio applications the most common problem is *latency*, the time delay from the user input to the corresponding sound output. In audio applications the output sound is usually calculated in blocks, i.e. a set of samples is calculated into a buffer and this buffer is then sent to the operating system's audio subsystem as a whole. Parameter values are often not allowed to change during one block. One block may be, e.g., 1ms long. The need for this approach arises because the system procedure that is used to send audio to the output usually has some per block overhead, which would cause problems if the samples were sent one at a time.

In multiprocessing environments, the operating system has its own FIFO (First In, First Out) sound buffer. This buffer is emptied from the other end by the audio subsystem with the audio sample rate, and filled by the application program block by block. The application program must be able to monitor the state of the output buffer so that the buffer is never completely emptied before new samples are written; if this condition is not satisfied, clicks or breaks occur. The buffer must therefore be filled so that if the system scheduler hands the control to another process, the control will return before the audio queue is empty. Furthermore, the buffer must not be too long because the maximum latency caused by the buffering depends linearly on the maximum buffer length. The total latency of the implemented guitar model is around 20 ms on a SGI O2 workstation, when not using real-time priorities that are possible only with superuser privileges.

The measurement of the total latency is a difficult task. The time measurement and logging systems provided by the operating systems themselves usually interfere with the system to be analyzed. Freed *et al.* (1997) have described an event and audio logging technique that uses an affordable multichannel digital audio recorder. In the system, special hardware solutions are used to convert network and MIDI bus events into audio signals recorded simultaneously with the actual audio output of the audio synthesis or processing system. Latencies can then be analyzed by comparing the audio tracks.

5 Synthesis Model Control

Physical modeling based sound synthesis methods have widely been demonstrated using separate notes rather than musical passages as examples. This is understandable, because musical, dynamic control of the synthesis methods is not easily achieved. Dynamic control poses various problems on all levels of sound synthesis. Both DSP-level methods and higher level interfaces are affected. If the dynamic control is also to be real-time, the efficiency issues will start to play a major role.

DSP-level problems are most often related to the fact that the parameters of the model change constantly. Therefore special care must be taken to avoid unwanted transients and discontinuities in the synthesized sound. Different parameter interpolation methods need to be applied for different parameters.

Higher-level control problems are related mainly to the musical control issues. To be useful to a musician, the synthesis method must have parameters that are musically meaningful. Those parameters must also be easily and versatilely controllable by some control method, e.g., using a keyboard. The lack of control capabilities of the standard controllers has turned some scientists to alternative controllers (see Cook, 1992). Jánosy *et al.* (1994) have used standard controllers and presented new “intelligent” control interfaces between the MIDI keyboard and a plucked string synthesis engine to make playing the guitar on the keyboard feasible. Laurson *et al.* (1999) have described a notation system that can be used to add expressional information to standard music notation and to control sound synthesis using this information.

In the following subsections, the issues relating to dynamic, real-time synthesis parameters and currently available synthesis model control protocols are presented. In section 5.5 a new control protocol and its application to the implemented guitar model is described.

5.1 DSP Parameters of the Guitar Model

The guitar model implementation introduced in chapter 4 contains an extensive number of signal processing parameters that together affect the resulting sound output. In the following subsections the excitation signal and adjustable parameters, and their influence on the model output are discussed. For information about controlling the parameter values, see section 5.5.

5.1.1 Excitation Signals

The string model and body resonator excitation signals determine the basic sound characteristics of the guitar model. By changing the signals it is possible to synthesize different playing techniques and different instruments altogether, e.g., the model can be applied to mandolin synthesis if appropriate excitation signals are available.

Excitation signal computation and equalization scheme based on sinusoidal modeling has been described by Tolonen (1998), and is used as such for the signals used in this work. Excitation signals are loaded from disk at initialization time and there can be different signals for every fret on each string. Also different playing styles may have differing excitations.

5.1.2 String Parameters

The dual-polarization string model presented in section 3.2 has quite a large number of adjustable parameters. The model consists of two simple string models for vibration polarization modeling, two additional filters, and a few extra multipliers. The total number of string model parameters excluding the coupling coefficients is twelve.

The parameters for the vibration polarizations include the real-valued delay line length and the coefficients of the loop filter. The delay line length determines the fundamental frequency as discussed in section 3.1.4. The loop filter is of first-order allpole type, and thus has two coefficients, which make it possible to control the zero-frequency gain and the high-frequency attenuation of the filter.

Varying the vibration polarization parameters affects the tonal quality of the decay part of a guitar tone. The audible duration of a guitar tone after plucking is determined by the loop filter gain, and the string material’s frequency-dependent damping is controlled by the loop filter frequency response. Currently, the only way to attenuate a note is to change the loop filter properties; another possible method for damping the string might be to utilize the amplitude-limiting nonlinearity model described in section 3.3.2. Naturally, the parameters of the two polarizations can be set independently of each other, e.g., two-stage attenuation can be implemented by setting the gains of the two loop filters to different values.

The string model excitation can be filtered according to desired tone quality with the timbre control filter and the pluck position filter. Filtering the excitation affects also the attack part of the instrument sound, unlike to the vibration polarization loop filter. Similarly to the loop filter, the timbre control filter is a first-order allpole filter with two coefficients for sound gain and brightness. The usage of the timbre control filter for expressive synthesis was also briefly described in section 3.5.2. The pluck position filter (see section 3.5.1) also has two parameters: the delay line length and the attenuation coefficient. The delay line length is directly proportional to the distance from the bridge to the plucking position and the attenuation coefficient c is usually $1 - \varepsilon$, where ε is a small positive number.

The dual-polarization string model also contains two mixing parameters that can be used to control the contribution of the two vibration polarizations to the sound. The input mixing coefficient corresponds to the plucking angle discussed in section 2.2.1 in the context of apoyando and tirando strokes. If the input mixing coefficient is near 0.0, the vertical vibration, i.e., the vibration perpendicular to the soundboard, will be dominant; correspondingly, for values near 1.0 the horizontal polarization will receive most of the vibration energy. Notice, that setting the value to 1.0 actually changes the instrument configuration so that the vertical polarization becomes a resonance string with no input other than via the sympathetic coupling input (Tolonen, 1998).

The output mixing coefficient determines the amount of energy transfer from each of the vibration polarization to the output sound. This makes it possible to model the directional impedance of a guitar bridge. The normal way is to use a value slightly less than 0.5, this makes the model natural in a sense that the vertical polarization drives the guitar bridge more efficiently than the horizontal polarization.

Guitar string sound is also affected by the strength of the sympathetic coupling phenomenon. Because no usable methods are known for approximating the coupling coefficients between the strings, manual tweaking of these parameters is needed. It is physically motivated to have coupling coefficients considerably less than 1. The coupling coefficient matrix used in the current implementation is

$$\mathbf{C} = \begin{bmatrix} 0.020 & 0.010 & 0.010 & 0.005 & 0.005 & 0.005 \\ 0.010 & 0.020 & 0.010 & 0.010 & 0.005 & 0.005 \\ 0.010 & 0.010 & 0.020 & 0.010 & 0.010 & 0.005 \\ 0.005 & 0.010 & 0.010 & 0.020 & 0.010 & 0.010 \\ 0.005 & 0.005 & 0.010 & 0.010 & 0.020 & 0.010 \\ 0.005 & 0.005 & 0.005 & 0.010 & 0.010 & 0.020 \end{bmatrix} . \tag{5.1}$$

5.1.3 Body Resonator Parameters

The two shared body resonators are implemented as second-order filters with five filter coefficients (see equation 3.22). Because the filter coefficients as such are not meaningful parameters for the body resonators, they are never adjusted directly. The adjustable parameters of the resonators are the center frequency and the bandwidth, and the filter coefficients are given by the formulae in eq. 3.23.

If needed, the center frequency f_{n0} and the bandwidth Δf_n may be calculated from the filter coefficients b_0 and a_1 as

$$f_{n0} = \frac{1}{2\pi}\arccos\left(\frac{a_1}{2(b_0 - 1)}\right) \tag{5.2}$$

and

$$\Delta f_n = \frac{1}{\pi}\arctan\left(\frac{b_0}{1 - b_0}\right). \tag{5.3}$$

5.2 Dynamic Parameters

If the parameters of the synthesis model are dynamically changed, the output signal of the model may suffer two kinds of consequences (Välimäki, 1995):

1. There may be a *transient* in the output signal. Transients occur if the filter state variables contain intermediate results that are related to the former filter coefficients, or if the state variables are cleared. The transient problem does not exist in correctly implemented nonrecursive filters, but dynamic recursive implementations have to take transients into account.
2. *Discontinuities* occur in both recursive and nonrecursive signal processing structures because, after the parameter update, the output signal is a result of a different filtering operation. Discontinuities may also cause audible artefacts to the synthesized sound.

Both the transients and the discontinuities can be reduced using various interpolation techniques. Specific techniques to suppress the transients of the recursive filter structures are also available.

5.2.1 Interpolation

Both the transient and the discontinuity of a filter output depend on the input signal and the magnitude of the coefficient change. It is possible to decrease the seriousness of these unwanted signal components by gradually interpolating the filter coefficients to the target values over a reasonably long transition time. This is because a small change in the coefficient values produces both a smaller transient and discontinuity than a large change (Zetterberg and Zhang, 1988).

The most straightforward type of parameter interpolation between two values is linear interpolation. In audio applications, also exponential interpolation, that is linear on logarithmic scale, may be desired. Techniques that take into account more than one previous value of the parameter can be used. For such techniques see a textbook in numerical analysis, such as (Burden and Faires, 1993).

Straightforward interpolation techniques may cause problems if used carelessly. If recursive filter structures are used, interpolating the parameters independently may lead to unstable filters, even if the coefficient values at the beginning and at the end result in a stable filter. In addition, interpolating the filter coefficients may not represent well the desired modifications of the filter responses. These problems can be tackled by interpolating the design parameters of the filter instead of the filter coefficients. For example, if the body resonance filter parameters are to be changed using some interpolation technique, the right way is to interpolate the center frequency and the bandwidth of the filter; the actual filter coefficients are always given by equation 3.23.

5.2.2 Cross-Fading Method

In cross-fading, two filters are run simultaneously, one with the old filter coefficients and one with the new ones. The filter with new coefficients is taken into use by grad-

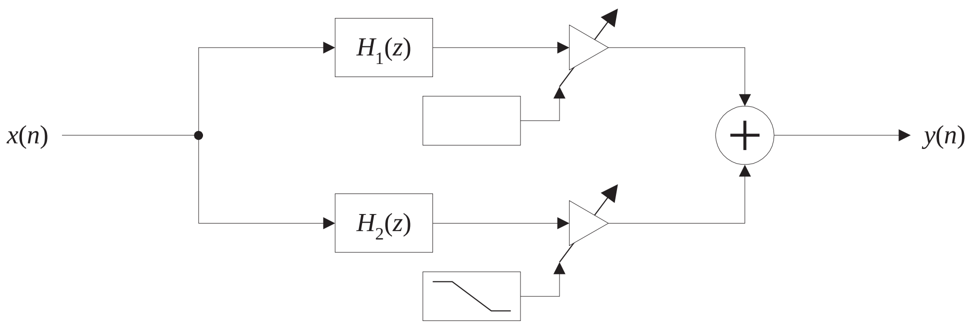

Figure 5.1: Cross-fading between two filter outputs.

ually increasing the input or output gain of the filter. Simultaneously, the old filter is faded out by decreasing its gain value, hence the name cross-fading. Figure 5.1 illustrates the procedure. Given sufficiently long transition time, the cross-fading method is able to eliminate both the discontinuities and the transients. An example of the cross-fading method is the legato cross-fading described by Jaffe and Smith (1995).

5.2.3 Transient Suppression

Välimäki and Laakso (1998) have described a novel method for transient suppression in recursive filters. Their approach is a modification of the *state-variable update technique* first introduced by Zetterberg and Zhang (1988). The Zetterberg–Zhang (ZZ) method is able to completely eliminate the transient related to a parameter change if all the past input samples are available. This requirement makes the method impractical, but the requirement can be loosened, if total elimination of the transient is not required (Välimäki and Laakso, 1998).

The key observation in the ZZ method is that the transient is eliminated if the state vector of a digital filter is modified at coefficient change time so that it corresponds to the steady state of the filter after coefficient change. But the steady state of a filter cannot be calculated if all the past input samples are not known.

The idea in the transient suppression method by Välimäki and Laakso (1998) is to run a transient eliminator filter parallel to the signal filter during N_a samples before the coefficient change at time index $n = n_c$. At the coefficient change time the state of the transient eliminator is copied to the signal filter. Figure 5.2 shows the different stages of the method.

The motivation to use a finite number of samples before coefficient change for transient suppression comes from the fact that the impulse response of a stable recursive filter decays exponentially and can thus be regarded to be of finite length for the application (Välimäki and Laakso, 1998). The number of observably nonzero values of the impulse response is called *effective length* of the impulse response. Estimating the effective length can be done, e.g., with an energy-based method proposed by Laakso and Välimäki (1999). Given effective length N_P and filter order N, the required advance time N_a is given by (Välimäki and Laakso, 1998)

$$N_a = N_P + N . \tag{5.4}$$

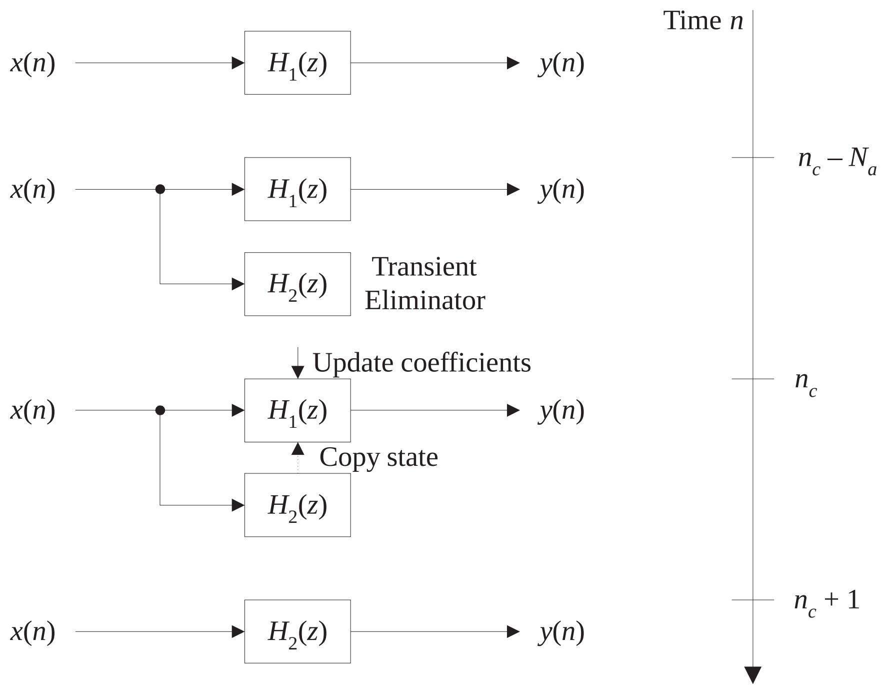

Figure 5.2: The principle of transient elimination. The transient eliminator filter is run for N_a samples before coefficient change in parallel with the signal filter. At time n_c the coefficients are updated and the state vector of the transient eliminator is copied to the signal filter. After (Välimäki and Laakso, 1998).

5.2.4 A Hybrid Method for Dynamic Delay Line Length

In the guitar model the length of the string delay line is the parameter with most severe problems, if the dynamic effects of parameter value updates are not handled carefully. In addition, the length may be altered by both continuous and discrete processes. The change associated with vibrato (Laurson *et al.*, 1999) and tension modulation (Tolonen *et al.*, 1999) is continuous in nature and may be automatically created. Continuous changes occur at regular intervals and are usually relatively small in value. Changing the sounding length of the string and the sound pitch by pressing the string against a fret causes an abrupt change in delay line length. These discrete changes occur only at special control events and may be quite large.

To handle the different requirements posed by different delay line length updates, the guitar model uses a hybrid approach that combines the transient suppression method with cross-fading. The method has two phases: 1) before the delay line length change the fractional delay filter with new coefficients is initialized as defined in the transient suppression method, and 2) the outputs of the two filters are cross-faded during a desired time period. The first phase enables suppression of transients, and the second phase eliminates discontinuities.

The first phase makes it possible to use very short or even zero cross-fading times with small changes in length, while the latter gives the possibility to use cross-fading that is long enough for avoiding troublesome discontinuities. A similar scheme has been proposed by Van Duyne *et al.* (1997). They used a special crossfading function that allows the fractional delay filter with the new coefficients to reach its steady state before the output signal is used.

5.3 Real-Time Control

Real-time implementation of the guitar model poses some extra requirements for the control performance of the model. Processing of incoming control events must be efficient, so that a sufficient number of control events can be processed without causing problems to the sound output. Also methods that require some kind of a look-ahead possibility are not realizable.

Each string of the guitar model has twelve parameters, and six of these are closely related to physical playing parameters assiciated with each newly plucked guitar sound, namely the lengths of the horizontal and vertical delay lines (2 parameters), the coefficients of the timbre control filter (2), the plucking position (1), and the input mixing coefficient (1). These are the parameters that may have to be set for each note played on the guitar model. For the other parameters suitable default values as a function of these playing variables may be found. The loop filter parameters mainly depend on the guitar and string properties and the length of the string, the attenuation coefficient of the plucking position filter depends on the plucking position and string length, and the output mixing coefficient is dependent on the properties of the instrument.

It can be assumed that the two delay line lengths can be changed in one control event, that the coefficients of the timbre filter require one event, and that the plucking position and input mixing coefficients are transmitted in the excitation event. The damping of a sounding note requires setting the total of four loop filter parameters, and resetting these parameters for undamped strings is also required. If all the loop filter parameters can be contained in one event, then exciting and damping one note will require five control events in total. In a case of quite a dense musical structure with twelve notes per second a total of 60 control events per second are needed. If all the aforementioned parameters are set using separate control events, 180 events per second are required.

In addition to the parameter changes related to note events, the lengths of the horizontal and vertical delay lines may be changed at a constant rate by, e.g, a vibrato control process. For musically relevant modulation frequencies in the range below 10 Hz, length update frequency of 100 Hz for one string should be enough. Assuming a six string guitar will raise the number of control events per second up to 780. Some special cases, such as rapid glissandos, may require even greater number of control events per second.

The time required to process the control events must be so small that the synthesis algorithm always has enough time to calculate the output sound samples before the output buffer of the audio subsystem runs out of new samples. In practice, it is usually not wise to aim for perfect worst-case performance, but to optimize the implementation for demanding material in the normal range. The implementation can also be made tunable so it can be optimized for different platforms and situations.

One more restriction is that in real-time applications there is no look-ahead possibility. This means that, e.g., the changes in parameter values cannot be prepared beforehand, as the transient suppression method described in section 5.2.3 would require. The parameter change may be delayed to compensate the unrealizable advance time. The other possibility is to always store a sufficient number of past samples so that at the parameter chance time, the filter structure can be initialized using the extra samples. This approach has been used in the delay line change method described in section 5.2.4.

5.4 Summary of Current Control Protocols

The MIDI protocol was the first and is the only widely supported real-time protocol to enable communication between two hardware devices, made to process musical information. It enabled professionals and amateurs to explore the possibilities of synthesized sound. The shortcomings of MIDI have later led to a number of proposals of more advanced control protocols. In the following subsections, current and proposed sound synthesis control protocols and schemes are described.

5.4.1 MIDI

MIDI is an acronym for Musical Instrument Digital Interface, and it originated as a real-time hardware communication protocol. The first commercial application was in 1986 and the protocol specification was published in 1988 (Hewlett *et al.*, 1997). The protocol is oriented to make information obtained from an electronic keyboard available to other devices. MIDI is very widely supported and is available for almost all computer platforms.

The MIDI control stream consists of consecutive messages that occur one at a time. The most basic musical information: the note pitch and duration are represented by note numbers and series of Note_On and Note_Off messages. The note numbers are directly related to the keys of the keyboard and the note messages relate to the “on” and “off” times of the keys. Each of the MIDI’s standard messages consists of a fixed number of bytes except for the variable length System-Exclusive (SysEx) message. This is why it is impossible to transmit any additional, instrument-specific information directly within the standard messages.

The addressing scheme in MIDI is based on MIDI channels. Each MIDI bus is capable of driving sixteen independent channels. The first byte of every MIDI message is a status byte that tells the type and possible channel number of the message. The messages with channel number are called Voice messages, and messages without channel number are either System Common or System Realtime messages. The second level of MIDI’s addressing hierarchy is the note number, which directly corresponds to the desired pitch of the instrument. This scheme makes the expression of some musical situations awkward, e.g., the pitch of a note might change over time or there might be two notes with the same pitch played simultaneously on one instrument.

One of the main restrictions of MIDI is the small number of possible channels. Also the direct cabling of input and output devices hinders construction of flexible setup of more than a few MIDI instruments. The second obvious restriction is the keyboard-orientedness of MIDI. The protocol integrates the timing, pitch, and loudness information into an indivisible Note_On message. For other instruments than the keyboard instruments, these parameters can, however, be independent. In the guitar, the right hand generally determines loudness and timing, whereas the left hand is responsible for pitch control. Further restrictions of the MIDI protocol that affect, e.g., the control intimacy and the ability to capture musical gestures, have been described by Moore (1988).

5.4.2 ZIPI

The ZIPI protocol was developed at the University of California, Berkeley as a more advanced alternative to MIDI, and it was first described in 1994 (McMillan, 1994). The ZIPI proposal included a new hardware specification as well as an application layer for transferring musical information across a ZIPI network. As the protocol never was widely adopted, it is safe to say that ZIPI is history (Freed, 1997), but the application layer and especially the Music Parameter Description Language (MPDL) are still worth a short description here.

Much of the MPDL design considers minimal usage of the limited transmission bandwidth. The growth of the network bandwidth has later made these aspects of the MPDL somewhat outdated. In the following discussion, the information coding issues are thus ignored, and the logical aspects of the language are emphasized.

To overcome the problems that arise in MIDI because the note number determines also the pitch of the note, in MPDL the note numbers are just identifiers given to the individual notes. MPDL has a three-level address hierarchy; *notes* are within *instruments*, which in turn are grouped to *families*. Any control signal can be applied to any level of the address hierarchy.

The parameters on the hierarchy levels are in principle separate and can interact with four different ways to affect the actual parameters used by the synthesis algorithm. The rules are “and”, “multiply”, “add”, and “overwrite”. The “and” rule is used only for the triggering messages, and describing it is beyond the scope of this text. If a parameter belongs to the “multiply” group, each layer of the hierarchy stores its most recent value for the parameter, and the final parameter value is a product of these. The “add” rule is similar to the “multiply” rule, except that the values are added together, not multiplied. The “overwrite” rule results in a direct substitution of the current parameter value, i.e., a parameter update on the instrument level overwrites the parameter values of each of the active notes of that instrument. (McMillan *et al.*, 1994)

To achieve efficient usage of network capacity, MPDL uses eight-bit ID numbers for parameters, instead of using, e.g., parameter names. The published parameter tables contain a few dozens of different parameters, the rest left empty for future use. The proposed parameters include also some psychoacoustical parameters, such as brightness, roughness, and attack character. Two of the more advanced MPDL messages are worth mentioning, namely the possibility to send modulation and parameter segment information, and the ability to control the allocation priority of the individual notes.

5.4.3 Synthesis toolKit Instrument Network Interface

Synthesis toolKit Instrument Network Interface (SKINI) is a sound synthesis control language developed by Perry Cook. SKINI has been designed to be used as a control protocol in Cook's "Synthesis ToolKit" (Cook, 1996a). The SKINI language is highly compatible with MIDI, extending it incrementally. SKINI has been used, e.g., in the FAUST sonification system to control the sound synthesis algorithms (Weinstein and Cook, 1997). The main differences from MIDI and the motivations behind SKINI are the usage of text-based messages and floating point numbers (Cook, 1996b).

Text-based messages allow any system capable of producing formatted text output to generate SKINI. Also any system that can read strings and make conversions from strings to floating point and integer numbers can consume SKINI. Because of the textual form of the format, scripts can easily be devised to perform simple and advanced tasks on the musical information. Most importantly, SKINI is human-readable; the tokens are meaningful names for the things they stand for, and debugging the files and programs is much easier than when using binary file formats and data streams.

Double-precision floating point numbers are used in SKINI to represent parameter values. Only the numbers used as identifiers are integers. The value ranges typically correspond to the MIDI value ranges so that incoming MIDI integers can be passed on without value modifications. The extra precision (the fractional parts) provided by floating point numbers can be used, e.g, to specify fractional pitch, or ignored, if full MIDI compatibility is required.

5.4.4 Open SoundControl

Open SoundControl (OSC) is a sound control protocol independent of any transport-layer protocol. The assumptions made concerning the network transport layer are commonly fulfilled by all modern network technologies. OSC assumes that the network bandwidth is over 10 Mbit/s, that the data is delivered in packets, that multiple devices may be connected together in one network, and that the network layer provides a return address so that a response can be sent back to the device that sends a message (Wright and Freed, 1997).

The data in OSC protocol is binary and is aligned on 4-byte boundaries. Numeric data can use big-endian 32-bit or 64-bit integers or IEEE floating point numbers. Strings are padded with extra null characters to reach the 4-byte boundary, if necessary. OSC data stream consists of messages that in turn include the address and message names and the associated data. Messages can be collected to *bundles* that are time-tagged collections of messages or bundles (note the recursive definition).

The addressing scheme in OSC is a dynamic, hierarchical system. Each sound synthesis system that can be controlled using OSC defines its own address hierarchy, the protocol itself does not constrain the object structure. This approach avoids the limitations of all the protocols that rely on fixed length bit fields as addresses. There is, however, no guarantee that any two synthesizers will have consistent name spaces. Thus, what the other synthesizer calls "volume", the other might call "amplitude". In OSC, multiple destination addresses can be specified using wildcards for regular expression like pattern matching. If multiple addresses match the given address string, the message is forwarded to all relevant objects, just as if several messages had been sent.

An important feature of the OSC protocol is the possibility to request information of a device. Because the address space of any device may be dynamic, the address space can be explored using the query messages. Also information of the parameter types and values, as well as human-readable documentation can be requested.

5.4.5 Structured Audio in MPEG-4

MPEG-4 is a comprehensive digital media standard developed by the Moving Picture Experts Group (MPEG). The formal ISO/IEC designation of the MPEG-4 standard is ISO/IEC 14496 (see ISO/IEC, 1999 for an overview of the standard). For sound synthesis control, the MPEG-4 standard provides the Structured Audio tools, including two relevant description languages: SAOL (Structured Audio Orchestra Language) and SASL (Structured Audio Score Language) (ISO/IEC, 1998).

SAOL is a digital signal processing language that is suitable for describing arbitrary sound synthesis and control algorithms as part of standard MPEG-4 content stream. It is used to define an *orchestra* consisting of *instruments*. SAOL is not a synthesis method, but rather a way of describing any synthesis method (Scheirer and Vercoe, 1999).

In MPEG-4, scores that are used to control sound synthesis, are described in the SASL language. Scores are sequential sets of commands that describe the way in which the sound generation algorithms described in SAOL are used to produce sound. Commands invoke the specified instruments at desired times to contribute to the overall sound production. SASL control can range from simple sound effects to very complicated acoustical scenes.

SASL itself is a simple language, simpler than many other existing score languages. It consists of a series of note, control, tempo, and dynamical wavetable events, all affecting an orchestra defined in a SAOL stream. Control events can control the whole orchestra or a set of currently running instruments. They may change the value of a global variable or the value of a specific instrument's variable. All times in a score file are measured in beats, i.e., in musical time. The mapping of the musical time to the real time is controlled using tempo events. Wavetables can be dynamically created or destructed by *table* lines of the score file. This mechanism makes it possible, e.g., to dynamically supply new sound samples to be used in sound scene synthesis.

5.5 A New Control Protocol and Its Application to the Guitar Model

All control protocols described in section 5.4 provide some attractive features for synthesis model control. However, for reasons stated later none of them seemed to be the ideal choice for guitar model control. A new control protocol has therefore been implemented. Although the only implementation of the protocol is the guitar model, the protocol itself is designed to be generic and portable.

The protocol consists of control events which are formatted strings of ASCII characters. Each control event is of the form `address/operation[?x1[?x2]]`. The parts enclosed in square brackets are optional, and `x1` and `x2` denote operation parameter values. This syntax is motivated by the similarity with the familiar URL notation.

In the following subsections the parts of the control event are described in detail and the application of the protocol to the current guitar model is discussed. In section 5.5.4 the addition of network addresses to the protocol is presented, and in section 5.5.5 some of the possible shortcomings of the protocol are discussed and solutions to those are presented.

5.5.1 Addressing Mechanism

The addressing mechanism in the new protocol is a hierarchical system similar to the familiar URL and file system hierarchies. The address or path string is constructed of tokens delimited with slash characters (“`/`”), each slash representing a new level of hierarchy. The root of the hierarchy tree is represented by a single slash. The tokens in turn consist of one or more alphanumeric characters and/or a few special characters. The allowed characters are presented in table 5.1.

Using the proposed addressing system it is possible to define any suitable address hierarchy for any specific synthesizer. For example, it is possible to implement a synthesizer that is capable of synthesizing the sounds of all the instruments in a symphony orchestra. In this case it would be intuitive and beneficial to construct the synthesizer so that the address space represents the conventional division of the instruments into sections. The whole orchestra could be divided into string instruments, wind instruments, and percussion. The wind instruments could be further divided into woodwinds and brass instruments, and the strings could be divided into first and second violins, violas, cellos, and contrabasses, and so on. A sketch of a possible orchestra synthesizer control tree is depicted in figure 5.3.

The hierarchical character string based addressing mechanism requires considerably more processing capacity and time than the simpler bit-field addresses of, e.g., the MIDI and ZIPI protocols. However, the advantages of such a system are also clear. Each synthesizer may define the most suitable address hierarchy, the addresses may be human-readable, and the addressing capabilities never become outdated. Also creation of modular synthesizer systems, where implemented components can be reused, is made simple by the possibility of placing readily implemented components into the control tree.

Table 5.1: Allowed characters in protocol address and operation strings

allowed character	alphanum | mark
alphanum	letter | digit
mark	`- | _ | . | ! | ~ | * | ' | ( | )`

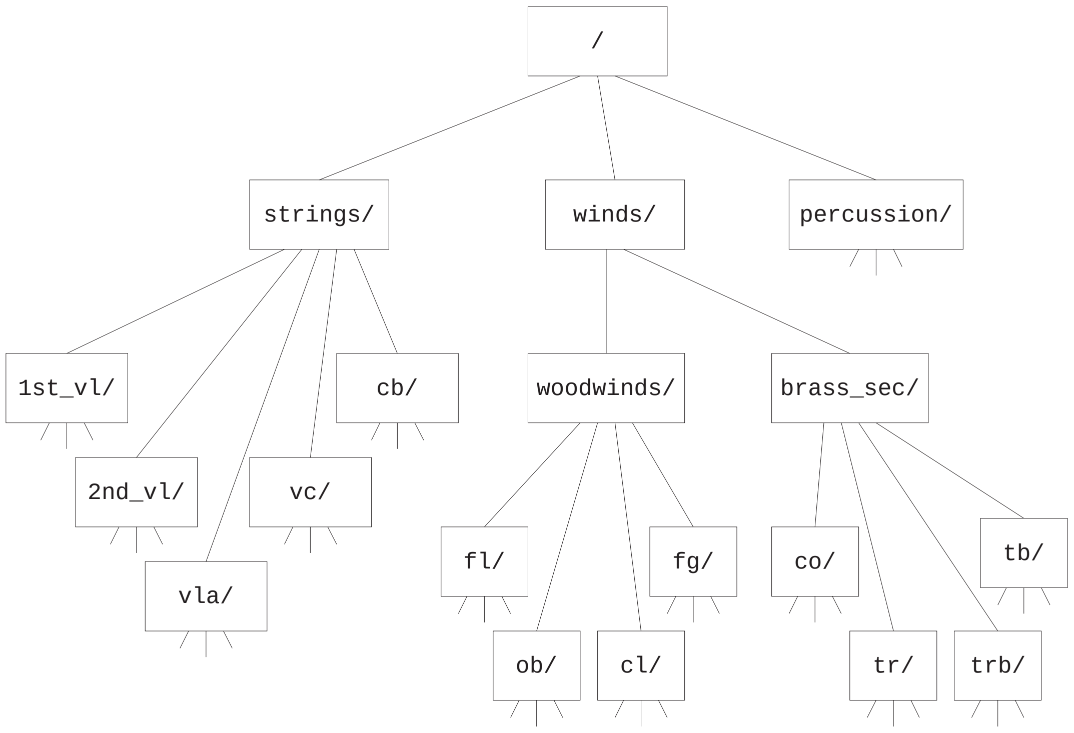

Figure 5.3: Control hierarchy of an orchestra synthesizer

5.5.2 Operations and Parameters

The `operation` part of a control event specifies the operations to be done on the object determined by the given `address`. The necessary parameters follow the operation in a 'question mark' separated list of values. The parameters are transmitted as ASCII characters and the transformation to the desired data type is done by the receiving unit. The characters allowed in operations and parameters are the same that are permissible in the event address (see table 5.1). Examples of different operations and their parameters are given in section 5.5.3, where the application of the control protocol to the guitar model is described.

To gain full benefit of the control model hierarchy, a few *reserved operations* with special meaning have been defined. These operations are special in a sense that when implemented on multiple hierarchy levels, they interact across the levels unlike ordinary operations, and that the names of the operations are reserved and may be used only for the predefined purposes. The reserved operations are shown in table 5.2 with the corresponding combination rules.

The `pitch` is given as a floating point number, where the integer part is the number of the nearest MIDI note below the desired pitch, and the fractional part is the upward offset. Thus pitch 60 is the middle C and pitch value 60.5 means a quarter tone sharp middle C. The `pitch` operation is reserved in a sense that it may not be used for any other purpose or with any other value range than specified. However, the protocol does not specify the interaction of `pitch` operations across several control levels.

Table 5.2: The reserved operations

Operation	Combination Rule
`pitch`	combine with `transpose`
`transpose`	add
`dynamics`	multiply
`amplitude`	multiply

The synthesizer may also implement the `transpose` operation on one or more control layers. The `tranpose` operation takes one numerical parameter that determines the amount of transposition. This means that if a `transpose` operation with parameter value 1 is sent to the ‘brass instruments’ level of an orchestra synthesizer (`/winds/brass_sec/transpose?1`), the result will be an upward transposition of all the notes played by the brass section by one semitone. The combination rule for transposition is add, i.e., if the winds level of the orchestra synthesizer is transposed one semitone down, and the brass section is transposed up by one semitone, the actual brass section sound will have the original pitch, but the woodwind section will sound one semitone flat.

The `dynamics` operation sets the musical dynamic value of the target object. Dynamics is given as a floating point number, value 1.0 corresponding to the chosen neutral musical dynamic level, mezzoforte (*mf*). The numerical dynamics values and the corresponding musical dynamic levels are presented in table 5.3. As with real instruments, the dynamics value may also affect the tonal properties of the synthesized sound. Dynamics values are combined across the hierarchy levels using multiplication, i.e., two

Table 5.3: Dynamic levels and `dynamics` values.

Dynamics	Dyn. Level
1/32	*pppp*
1/16	*ppp*
1/8	*pp*
1/4	*p*
1/2	*mp*
1	*mf*
2	*f*
4	*ff*
8	*fff*
16	*ffff*

mezzofortes give mezzoforte, mezzoforte and piano give piano, two mezzopianos give piano, and so on. The idea is to provide a "conductor's view" to musical dynamics. The user may use this system, e.g., to adjust the proportional dynamical levels of the sections of the orchestra synthesizer irrespective of the dynamical levels of the individual instruments.

The `amplitude` operation affects the volume of the sound, but without any modification to the sound's timbre. It has the same effect as setting the amplification level on a mixing console. Similarly to the dynamics operation, the combination rule for amplitude is multiply. The default value, i.e., the value before any control event, for both the dynamics and the amplitude is 1.0.

The implementation of the reserved operations on all hierarchy levels is not required, but is recommended whenever it is natural and feasible. The purpose of the reserved operations is to provide user some properties that are similar between different synthesis engines and to ensure that some concepts are used consistently across all implementations.

5.5.3 Guitar Model Application

The control hierarchy of the implemented guitar model is depicted in figure 5.4. As can be seen, the hierarchy is closely related to the guitar model structure presented in section 4.5.1. In the figure address hierarchy is represented by rectangular boxes and solid lines, and operations are represented by rounded boxes. The figure gives details only for the control of the guitar body model resonators; the string model control is elaborated later.

The guitar model is divided into body and individual string entities. The `body` subtree of the guitar model control hierarchy is further divided into `reson1` and `reson2` parts, corresponding to the first and the second body resonator, respectively. In addition, the `body` level has an `amplitude` operation, which may be used to set the body/

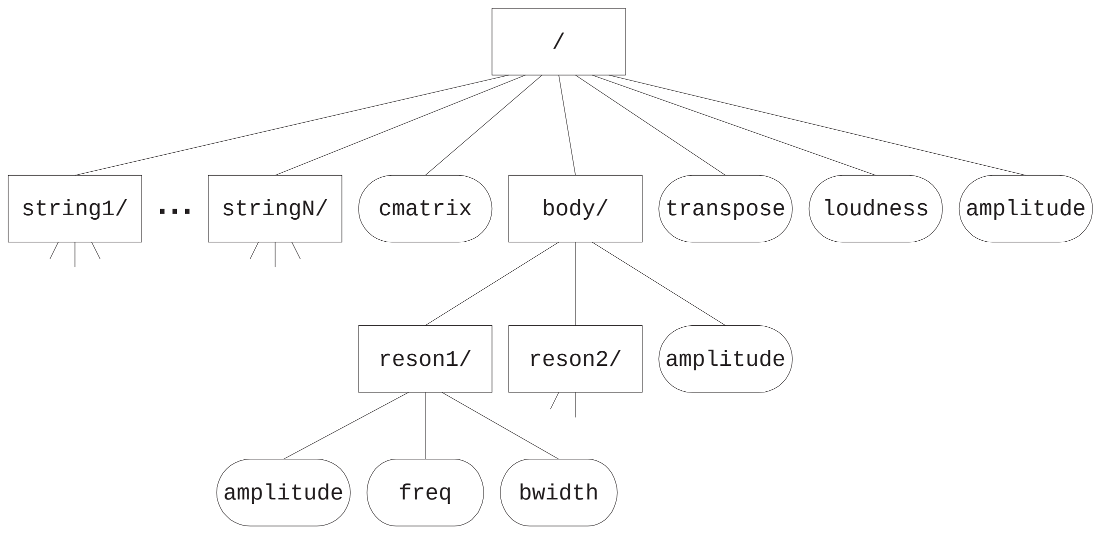

Figure 5.4: Guitar model control tree.

string balance of the instrument sound. The operations of the resonators include `freq`, `bwidth`, and `amplitude`, for center frequency, filter bandwidth, and resonance amplitude, respectively. Each of the body and resonator level operations take one floating point number as a parameter. The center frequency and filter bandwidth are given on normalized frequency scale, i.e., the user must be aware of the guitar model sampling rate when changing these parameters.

In addition to the body and string model subtrees, the control contains four operations directly under the root node. The `cmatrix` coupling operation implements the control of the coupling matrix **C** (see section 3.2.1), and gets three parameters: one integer number for each matrix index and one floating point number for the actual coupling coefficient value. The order of the parameters is: 1) source string number, 2) destination string number, and 3) the coefficient value. The `transpose`, `dynamics`, and `amplitude` operations belong to the reserved operations, and thus their meanings and properties are as discussed in section 5.5.2.

The control implementation of the dual-polarization string models is depicted in figure 5.5. The operations below the string level are placed inside one big rounded box instead of individual boxes, because of the large number of the operations. The operations below the individual string level mostly used for normal guitar model control during playing are `pluck`, `dynamics`, `pitch`, `loop_gain`, and `loop_shape`. The additional operations and address entities are for more advanced playing control, for configuration and for direct DSP-level control of the model. The operations of the vertical vibration mode (`vert/`) are the same as for the horizontal model (`horiz/`).

The `pluck` operation is used to actually excite the string, i.e., it sets different pointer values so that the proper excitation signal is fed to the string model during several subsequent samples. The pluck operation recognizes two optional parameters: the gain that multiplies the excitation signal and the index of the wavetable used as excitation signal. The gain is given as a floating point number and the wavetable index is an in-

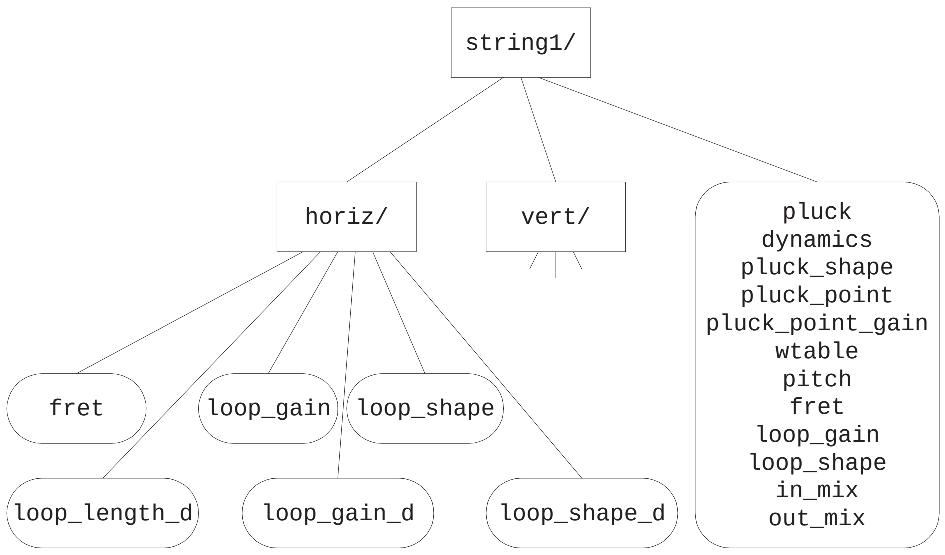

Figure 5.5: Control tree for a dual-polarization string model.

teger. The excitation gain can also be set using the `dynamics` operation, and the excitation wavetable can be changed by the `wtable` operation. If gain or wavetable values are given, they overwrite the values previously set by `dynamics` or `wtable`.

Properties of the excitation signal filtering can be set using the `pluck_shape`, `pluck_point`, and `pluck_point_gain` operations. The `pluck_shape` operation sets the frequency-dependent properties of the timbre control filter. This operation takes one floating point number as an argument, the range of the argument being [-1,1]. Negative values of the argument lead to lowpass characteristics, positive values result in high-frequency emphasis, and zero is the neutral value. The operations `pluck_point` and `pluck_point_gain` set the parameters of the pluck position filter described in section 3.5.1.

The `pitch` operation sets the length of the string delay lines so that the resulting sound output has the desired pitch. The `fret` operation is similar to the `pitch` operation, but whereas `pitch` sets the absolute pitch, the `fret` operation sets the pitch as an offset from the open pitch of the string. Both operations take one floating point value as an argument. The `pitch` and `fret` operations under the string level overwrite the values set using the `fret` operations under the `horiz/` and `vert/` entities.

The `loop_gain` and `loop_shape` operations are used to set the damping properties of the polarization loops. The `loop_gain` operation sets the zero-frequency gain of the loop filters, and the `loop_shape` operation sets the frequency-dependent damping. The influence of the `loop_gain` and `loop_shape` operations is based on using tables for default values of the related DSP-level parameters, and using the user-specified operation arguments as modifiers to the default values. For each string and each fret there exists tabulated values for both loop parameters. The actual DSP-level loop gain g is given by

$$g = g_d^{(1/u)}, \tag{5.5}$$

where $g_d \in [0.0, 1.0]$ is the default gain and u is the `loop_gain` value given by the controller. Similarly, the filter coefficient a is given by

$$a = -|a_d|^v, \tag{5.6}$$

where $a_d \in (-1.0, 0.0]$ is the default coefficient and v is the `loop_shape` argument. Similarly to the `fret` operation, the `loop_gain` and `loop_shape` operations of the string overwrite the values set using the corresponding operations of the individual vibration polarizations.

The operations `loop_length_d`, `loop_gain_d`, and `loop_shape_d` can be used to set the parameters of the vibration polarizations directly (hence the letter d). When these operations are used, the parameter mappings of equations 5.5 and 5.6 are not used, but the DSP parameters are immediately updated using the provided argument values.

The operations `in_mix` and `out_mix` are used to set the mixing coefficients m_p and m_o of the dual-polarization string model. The `out_mix` operation is mainly used for model configuration, whereas the `in_mix` operation can be used to set the string plucking angle. Both operations take one floating point argument in the range [0.0, 1.0].

The user may switch between different string model and the resonator excitation signals using the proposed protocol, but the properties of the excitation signals cannot currently be changed. It is suggested that a `load` operation be implemented for this purpose at a later time.

5.5.4 Using the Protocol Over a Network Interface

The proposed control protocol may be used over a network interface quite easily. The network address can be appended in front of the hierarchical address part of the protocol. If, e.g., an orchestra synthesizer software was to listen the port 10001 on host `synth.host.fi`, the complete address of the first violin section could be `synth.host.fi:10001/strings/1st_vl/`.

5.5.5 Discussion

The control protocol has similarities with several of the earlier sound synthesis control protocols. The `pitch` operation of the protocol uses the same value range for the musical pitches as the MIDI protocol (see section 5.4.1), but implementing the microtonal pitches similarly to the SKINI protocol (see section 5.4.3). The idea of the reserved parameters that interact across the hierarchy levels has been adopted from the ZIPI proposal (see section 5.4.2). The addressing mechanism of the proposed protocol is similar to the addressing scheme of the OSC control protocol (see section 5.4.4). However, the data is not binary as it is in the OSC protocol, and the network address inclusion is differently defined.

There are a few features, which might prove useful in the future, that are not included in the proposed control protocol. It is not possible to transmit multiple control events concurrently, and the events are not time-stamped.

Multiple concurrent events could be implemented by allowing a special meta control event that could include several messages in one entity. The OSC protocol reserves the string `#bundle` to indicate a bundle of events. Another mechanism for concurrent events in OSC is the possibility to use a pattern matching syntax similar to regular expressions to refer to multiple operations in a single event. Thus, address `/strings/*_vl` could refer to both first and second violins in the orchestra synthesizer example. Pattern matching is potentially a very efficient technique in reducing the network bandwidth needed for the protocol, but requires a substantially more complex implementation of the address parsing mechanism than does a protocol without pattern matching.

In the proposed protocol, the control events are not time-tagged, but the events take effect as soon as they are received. However, the traveling time of separate events may differ from each other so that time intervals between events are not the same for the receiver as they were when sent. This variation of the time intervals is called jitter. If the events included time tags, it would be possible to resynchronize the events with bounded latency, thus eliminating the jitter. The OSC protocol uses time tags with the time representation also used by the Network Time Protocol (NTP) timestamps (Wright and Freed, 1997).

It would have been possible to implement the proposed protocol as an OSC application to the guitar model. This way multiple concurrent events as well as time tags would have been easy to implement. The decision to use ASCII parameters and the implementation of the combinatory operations are however different from the approaches used in OSC so that the development of the new protocol was motivated.

6 Conclusions and Future Work

In this thesis, development, structure, implementation, and control of a model-based guitar synthesizer were discussed. The computational model of an acoustic guitar was already known to be able to produce good quality synthetic sound. Thus this work concentrated on implementing the model on a workstation platform, and on the control of the model so that natural-sounding guitar music synthesis can be achieved.

The synthesis model of this work was implemented using the C++ programming language and object-oriented programming paradigm for efficient digital signal processing. One of the main goals was to create reusable signal processing structures for model-based sound synthesis. The work was originally done on one hardware/software platform, but portability has been taken into account in the design.

Using the synthesis model for music synthesis required development of the control mechanisms of the model. Typical playing techniques of the acoustic guitar, and implementing some of them using the guitar model were described in this thesis. For synthesis control a new protocol was designed during the work.

The implemented guitar model with six dual-polarization string models and sympathetic couplings runs easily in real-time as was desired. The primary goal of creating a real-time guitar synthesizer as a combination of rather general purpose signal processing objects was thus satisfied. With properly implemented interpolation and transient suppression methods, the implementation is able to produce clean sound with no disturbing clicks. With the implemented control protocol, it is possible to control the parameters of the synthesis model using a structured addressing scheme. The implemented parameter mappings make these parameters more accessible to the user.

The rest of this chapter is divided into two parts. In section 6.1 a summary of the thesis is presented by briefly revising the contents of each chapter. Section 6.2 gives directions for future work in the field of real-time expressive model-based sound synthesis.

6.1 Thesis Summary

After the introduction to the thesis and to model-based sound synthesis in chapter 1, chapter 2 described the construction, acoustics, as well as the playing techniques of the acoustic guitar. The different playing techniques were addressed so that basic knowledge of the relationship of the physical phenomena to the instrument sound may be gained.

Chapter 3 discussed the development of the implemented guitar model from signal processing point of view. In the beginning of the chapter the modeling of vibrating strings with digital waveguide models was outlined. The waveguide model was formulated as a single delay line string model, that was then extended to a dual-polarization string model with sympathetic coupling. Models for two different kinds of nonlinearities of string vibration were also discussed. Guitar body modeling was addressed and both commuted body modeling and shared body resonators were introduced. Guitar model properties that permit synthesis of some expressive means by filtering the excitation signal were described, and finally, computationally efficient multirate model structures were discussed.

The real-time implementation of the guitar model was the subject of the fourth chapter. At first, the classes of the implemented DSP software library were presented from the abstract base classes to the filter, ring buffer, string model, and body resonator implementations. After the DSP library classes, the base class for instrument models was described. The structure of the implemented guitar model was depicted and the guitar model class was described. The chapter ended with a discussion of the signal processing capabilities of the modern workstation hardware and software.

The control problems of the real-time guitar synthesizer were discussed in chapter 5. DSP-level parameters of the guitar model were reviewed and their relevance to the sound quality of the model was described. The implications of dynamically altering the parameter values, as well as techniques used to eliminate undesired transients and discontinuities were presented. Some problems specific to a real-time implementation were also discussed.

The rest of the chapter 5 concentrated on the protocols used to control sound synthesizers. First, a summary of five currently available or proposed control protocols was given. Second, a new protocol for sound synthesis control was proposed, and its application to the guitar model was described. The chapter was concluded with a discussion of the similarities of the proposed protocol to the earlier ones, and of the shortcomings and possible improvements.

6.2 Future Work

The future development of expressive model-based synthesis will consist of both developing new model structures and developing new control mechanisms for the models. As the new structures will produce even more realistic instrument sounds, the new control mechanisms will provide more efficient, intuitive, and expressive musical control.

The synthesis model described in this work can yield high-quality guitar sound synthesis with moderate computational cost. Truly expressive music synthesis, however, requires a large number of precalculated excitation signals, that may have to be different for different playing techniques and styles. Thus the model with body resonator filters parallel to the string model may be impractical.

With modern computers, guitar models that could not be implemented as real-time applications a few years back, are today feasible. Future synthesis models may incorporate guitar body models implemented as digital filters, rather than precalculating the body response to the excitation signal. If the guitar body model can be excluded from the string model input signal, the interaction between the player and the instrument can be parameterized and the nonlinear effects that depend on the properties of the string vibration can be more easily and realistically reproduced.

The development of the control mechanisms for the physics-based sound synthesis models depends greatly on the development of the models themselves. The methods of producing expressive synthesis may differ from model to model, but generic frameworks and definition of desired features will be valuable.

The MIDI protocol has determined the manner of controlling sound synthesizers for quite some time now. The development of the synthesis methods and the growing demands of the users have pinpointed the defects of MIDI. In the future the control protocols will most likely be more application specific, or alternatively, specific applications of more generic control protocols.

References

Burden, R. L. and Faires, D. J. 1993. *Numerical Analysis, 5th Edition*. PWS Publishing Company, Boston, Massacusetts, USA, p. 768.

Campbell, P. 1978. *Juan Martín's Guitar Method, El arte flamenco de la guitarra*. United Music Publishers Ltd., London, UK.

Chaigne, A. 1991. Viscoelastic Properties of Nylon Guitar Strings, *Catgut Acoustical Society Journal*, 1(7), pp. 21–27.

Chaigne, A. 1992. On the Use of Finite Differences for Musical Synthesis; Application to Plucked Stringed Instruments, *Journal d'Acoustique*, 5(2), pp. 181–211.

Christensen, O. and Vistisen, B. B. 1980. Simple Model for Low-Frequency Guitar Function, *Journal of the Acoustical Society of America*, 68(3), pp. 758–766.

Cook, P. R. 1992. A Meta-Wind Instrument Physical Model, and a Meta-Controller for Real Time Performance Control, *Proceedings of the 1992 International Computer Music Conference (ICMC'92),* San Jose, California, USA.

Cook, P. R. 1996a. Synthesis ToolKit in C++ [online, referenced April 15 1999]. Available in PostScript format at `<http://www.cs.princeton.edu/˜prc/SKINI09.txt.html>`.

Cook, P. R. 1996b. Synthesis toolKit Instrument Network Interface [online, referenced April 15 1999]. Available in HTML format at `<http://www.cs.princeton.edu/˜prc/STKPaper.ps>`.

Cumpiano, W. R. and Natelson, J. D. 1993. *Guitarmaking: Tradition and Technology; A Complete Reference for the Design and Construction of the Steel-String Folk Guitar & the Classical Guitar*. Chronicle Books, San Francisco, California, USA, p. 388.

Duncan, C. 1980. *The Art of Classical Guitar Playing*. Summy-Birchard Music, Princeton, New Jersey, USA, p. 132.

Elejabarrieta, M. J. and Ezcurra, A. 1997. Material Dependence of the Vibrational Behaviour of the Guitar's Plate, *Proceedings of the Institute of Acoustics*, Vol. 19, pp. 143–148. Presented at the International Symposium on Musical Acoustics, Edinburgh, UK.

Fletcher, N. H. and Rossing, T. D. 1991. *The Physics of Musical Instruments*, Springer-Verlag, New York, USA, p. 620.

Freed, A. 1997. ZIPI Home Page [online]. Center for New Music and Audio Technologies, updated June 1997 [referenced April 13 1999]. Available in HTML format at `<http://www.cnmat.berkeley.edu/ZIPI/>`.

Freed, A., Chaudhary, A. and Davila, B. 1997. Operating Systems Latency Measurement and Analysis for Sound Synthesis and Processing Applications, *Proceedings of the 1997 International Computer Music Conference*, Thessaloniki, Greece, pp. 244–247.

Giertz, M. 1979. *Den klassiska gitarren, Instrumentet musiken mästarna*. (in Swedish) P. A. Norstedt & söners förlag, Stockholm, Sweden, p. 313.

Grunfeld, F. V. 1969. *The Art and Times of the Guitar, An Illustrated History of Guitars and Guitarists*. The Macmillan Company, Collier-Macmillan Canada Ltd., Toronto, Canada, p. 340.

Helminen, K. 1999. Personal communication.

Hewlett, W. B., Selfridge-Field, E., Cooper, D., Field, B. A., Ng, K.-C. and Sitter, P. 1997. MIDI, in Selfridge-Field, E. (editor) 1997. *Beyond MIDI: The Handbook of Musical Codes*, pp. 41–72, The MIT Press, Cambridge, USA.

ISO/IEC FCD 14496-3 Subpart 5, Information Technology - Coding of Audiovisual Object - Low Bitrate Coding of Multimedia Object, Part 3: Audio, Subpart 5: Structured Audio, May 1998.

ISO/IEC JTC1/SC29/WG11 N2725, Overview of the MPEG-4 Standard, March 1999, Seoul, South Korea. Also available in HTML format at <http://www.cselt.it/mpeg/standards/mpeg-4/mpeg-4.htm>.

Jackson, L. B. 1989. *Digital Filters and Signal Processing, 2nd Edition*. Kluwer Academic Publishers, p. 410.

Jaffe, D. A. and Smith, J. O. 1983. Extensions of the Karplus-Strong Plucked-String Algorithm, *Computer Music Journal*, 7(2), pp. 76–87.

Jaffe, D. A. and Smith, J. O. 1995. Performance Expression in Commuted Waveguide Synthesis of Bowed Strings, *Proceedings of the 1995 International Computer Music Conference (ICMC'95)*, Banff, Canada, pp. 343–346.

Jánosy, Z., Karjalainen, M. and Välimäki, V. 1994. Intelligent Synthesis Control with Applications to a Physical Model of the Acoustic Guitar, *Proceedings of the 1994 International Computer Music Conference*, Aarhus, Denmark, pp. 402–406.

Karjalainen, M. and Laine, U. K. 1991. A Model for Real-Time Sound Synthesis of Guitar on a Floating-Point Signal Processor, *Proceedings of the 1991 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'91)*, vol. 5, Toronto, Canada, pp. 3653–3656.

Karjalainen, M. and Smith, J. 1996. Body Modeling Techniques for String Instrument Synthesis, *Proceedings of the 1996 International Computer Music Conference*, Hong Kong, pp. 232–239.

Karjalainen, M. and Välimäki, V. 1993. Model-Based Analysis/Synthesis of the Acoustic Guitar, *Proceedings of the Stockholm Music Acoustic Conference*, Stockholm, Sweden, pp. 443–447.

Karjalainen, M., Välimäki, V. and Jánosy, Z. 1993. Towards High-Quality Sound Synthesis of the Guitar and String Instruments, *Proceedings of the 1993 International Computer Music Conference (ICMC'93)*, Tokyo, Japan, pp. 56–63.

Karjalainen, M., Välimäki, V. and Tolonen, T. 1998. Plucked String Models: from the Karplus–Strong Algorithm to Digital Waveguides and Beyond. *Computer Music Journal*, 22(3), pp. 17–32.

Karplus, K. and Strong, A. 1983. Digital Synthesis of Plucked-String and Drum Timbres, *Computer Music Journal*, 7(2), pp. 43–55. Also published in Roads, C. (editor) 1989. *The Music Machine*, pp. 467–479. The MIT Press, Cambridge, Massachusetts.

Laakso, T. I. and Välimäki V. 1999. Energy-Based Effective Length of the Impulse Response of a Recursive Filter, *IEEE Transactions on Instrumentation and Measurement*, 48(1), pp. 7–17.

Laakso, T. I., Välimäki, V., Karjalainen, M. and Laine, U. K. 1996. Splitting the Unit Delay—Tools for Fractional Delay Filter Design, *IEEE Signal Processing Magazine*, 13(1), pp. 30–60.

Laurson, M., Hiipakka, J., Erkut, C., Karjalainen, M., Välimäki, V., Kuuskankare, M. and Takala, T. 1999. From Expressive Notation to Model-Based Sound Synthesis: a Case Study of the Acoustic Guitar, *Proceedings of the 1999 International Computer Music Conference (ICMC'99)*, Beijing, China.

Legge, K. A. and Fletcher, N. H. 1984. Nonlinear Generation of Missing Modes on a Vibrating String, *Journal of the Acoustical Society of America*, 76(1), pp. 5–12.

Lippman, S. B. 1991. *C++ Primer, 2nd Edition*. Addison-Wesley, p. 614.

McMillen, K. 1994. ZIPI: Origins and Motivations, *Computer Music Journal*, 18(4), pp. 47–51.

McMillen, K., Wessel, D. L. and Wright, M. 1994. The ZIPI Music Parameter Description Language, *Computer Music Journal*, 18(4), pp. 52–73.

Meyer, J. 1983a. Quality Aspect of the Guitar Tone, in Jansson, E. V. (editor) *Function, Construction and Quality of the Guitar: Papers given at a seminar organized by the Committee for the Acoustics of Music*, Publications issued by the Royal Swedish Academy of Music, No. 38, pp. 51–75, Stockholm, Sweden.

Meyer, J. 1983b. The Function of the Guitar Body and Its Dependence Upon Constructional Details, in Jansson, E. V. (editor) *Function, Construction and Quality of the Guitar: Papers given at a seminar organized by the Committee for the Acoustics of Music*, Publications issued by the Royal Swedish Academy of Music, No. 38, pp. 77–100, Stockholm, Sweden.

Middleton, R. 1997. *The Guitar Maker's Workshop*. The Crowood Press Ltd., Ramsbury, UK, p. 160.

Moore, F. R. 1988. The Dysfunctions of MIDI, *Computer Music Journal*, 12(1), pp. 19–28.

Oppenheim, A. V., Willsky, A. S. and Young, I. T. 1983. *Signals and Systems*. New York, Prentice–Hall, p. 796.

Orfanidis, S. J. 1996. *Introduction to Signal Processing*. Prentice–Hall, Englewood Cliffs, New Jersey, USA, p. 798.

Oribe, J. 1985. *The Fine Guitar*. Mel Bay Publications, Inc., Pacific, USA, p. 96.

Pavlidou, M. and Richardson, B. E. 1997. The String-Finger Interaction in the Classical Guitar: Theoretical Model and Experiments, *Proceedings of the Institute of Acoustics*, Vol. 19, pp. 55–60. Presented at the International Symposium on Musical Acoustics, Edinburgh, UK.

Proakis, J. G. and Manolakis, D. G. 1992. *Digital Signal Processing; Principles, Algorithms, and Applications. 2nd edition*. Macmillan Publishing Company, New York, USA, p. 969.

Rabiner, L. R. and Gold, B. 1975. *Theory and Application of Digital Signal Processing*, Prentice–Hall, Englewood Cliffs, New Jersey, USA, p. 762.

Rank, E. and Kubin, G. 1997. A Waveguide Model for Slapbass Synthesis, *Proceedings of the 1997 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'97)*, vol. 1, Munich, Germany, pp. 443–446.

Redgate, 1998. About Redgate Guitars [online]. Jim Redgate Guitars, 1998, updated November 10 1998 [referenced March 18 1999]. Available in HTML format at <`http://www.ozemail.com.au/˜redgate/index.html`>.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen W. 1991. *Object-Oriented Modeling and Design.* Prentice–Hall International, 1991.

Russell, D. 1988. Classical Technique, in *Stimpson, M. (editor) 1988. The Guitar; A Guide for Students and Teachers*, Oxford University Press, New York, USA, pp. 142–157.

Savioja, L., Rinne, T. and Takala, T. 1994. Simulation of Room Acoustics with a 3-D Finite Difference Mesh, *Proceedings of the 1994 International Computer Music Conference (ICMC'94)*, Aarhus, Denmark, pp. 463–466.

Scheirer, E. D. and Vercoe, B. L. 1999. SAOL: The MPEG-4 Structured Audio Orchestra Language, *Computer Music Journal*, 23(2), pp. 31–51.

Smith, J. O. 1985. A New Approach to Digital Reverberation Using Closed Waveguide Networks, *Proceedings of the 1985 International Computer Music Conference (ICMC'85)*, Vancouver, Canada, pp. 47–53.

Smith, J. O. 1987. *Music Applications of Digital Waveguides*, Technical Report STAN-M-39, CCRMA, Department of Music, Stanford University, USA, p. 181.

Smith, J. O. 1991. Viewpoints on the History of Digital Synthesis, *Proceedings of the 1991 International Computer Music Conference (ICMC'91)*, Montreal, Canada, pp. 1–10.

Smith, J. O. 1992. Physical Modeling Using Digital Waveguides, *Computer Music Journal*, 16(4), pp. 74–91.

Smith, J. O. 1993. Efficient Synthesis of Stringed Musical Instruments, *Proceedings of the 1993 International Computer Music Conference (ICMC'93),* Tokyo, Japan, pp. 64–71.

Taylor, J. 1978. *Tone Production on the Classical Guitar*. Musical New Services Ltd., London, UK, p. 80.

Tolonen, T. 1998. *Model-Based Analysis and Resynthesis of Acoustic Guitar Tones*. Master's thesis. Helsinki University of Technology, Laboratory of Acoustics and Audio Signal Processing, Report 46, Espoo, Finland, p. 109. Also available at `<http://www.acoustics.hut.fi/˜ttolonen/thesis.html>`.

Tolonen, T., Välimäki, V. and Karjalainen, M. 1998a. *Evaluation of Modern Sound Synthesis Methods*. Helsinki University of Technology, Laboratory of Acoustics and Audio Signal Processing, Report 48, Espoo, Finland, p. 114. Also available at `<http://www.acoustics.hut.fi/˜ttolonen/sound_synth_report.html>`.

Tolonen, T., Välimäki, V. and Karjalainen, M. 1998b. A New Sound Synthesis Structure for Modeling the Coupling of Guitar Strings, *Proceedings of the IEEE Nordic Signal Processing Symposium (NORSIG'98),* Vigsø, Denmark, pp. 205–208.

Tolonen, T., Välimäki, V. and Karjalainen, M. 1999. Modeling of Tension Modulation Nonlinearity in Plucked Strings, accepted for publication in *IEEE Transactions on Speech and Audio Processing*.

Tyler, J. 1980. *The Early Guitar, A History and Handbook*. Early Music Series 4. Oxford University Press, London, UK, p. 176.

Välimäki, V. 1995. *Discrete-Time Modeling of Acoustic Tubes Using Fractional Delay Filters*. Doctoral thesis. Helsinki University of Technology, Laboratory of Acoustics and Audio Signal Processing, Report 37, Espoo, Finland, p. 193. Also available at `<http://www.acoustics.hut.fi/~vpv/publications/vesa_phd.html>`.

Välimäki, V., Huopaniemi, J., Karjalainen, M. and Jánosy, Z. 1996. Physical Modeling of Plucked String Instruments with Application to Real-Time Sound Synthesis, *Journal of the Audio Engineering Society*, 44(5), pp. 331–353.

Välimäki, V. and Laakso, T. I. 1998. Suppression of Transients in Variable Recursive Digital Filters with a Novel and Efficient Cancellation Method. *IEEE Transactions on Signal Processing*, 46(12), pp 3408–3414.

Välimäki, V. and Takala T. 1996. Virtual Musical Instruments—Natural Sound Using Physical Models, *Organised Sound*, 1(2), pp. 75–86.

Välimäki, V. and Tolonen, T. 1997. Multirate Extensions for Model-Based Synthesis of Plucked String Instruments, *Proceedings of the 1997 International Computer Music Conference (ICMC'97),* Thessaloniki, Greece, pp. 244–247.

Välimäki, V. and Tolonen, T. 1998. Development and Calibration of a Guitar Synthesizer, *Journal of the Audio Engineering Society*, 46(9), pp. 766–778.

Välimäki, V., Tolonen, T., and Karjalainen, M. 1998. Signal-Dependent Nonlinearities for Physical Models Using Time-Varying Fractional Delay Filters, *Proceedings of the 1998 International Computer Music Conference (ICMC'98)*, Ann Arbor, Michigan, USA, pp. 264–267.

Välimäki, V., Tolonen, T. and Karjalainen, M. 1999. Plucked-String Synthesis Algorithms With Tension Modulation Nonlinearity, *Proceedings of the 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'99)*, Phoenix, Arizona, vol. 2, pp. 977–980.

Van Duyne, S. A. and Smith J. O. 1993. Physical Modeling with the 2-D Digital Waveguide Mesh, *Proceedings of the 1993 International Computer Music Conference (ICMC'93)*, Tokyo, Japan, pp. 40–47.

Van Duyne, S. A., Jaffe, D. A., Scandalis, G. P. and Stilson, T. S. 1997. A Lossless, Click-free, Pitchbend-able Delay Line Loop Interpolation Scheme, Proceedings of the 1997 International Computer Music Conference (ICMC'97), Thessaloniki, Greece, pp. 252–255.

Weinreich, G. 1977. Coupled Piano Strings, *Journal of the Acoustical Society of America*, 62(6), pp. 1474–1484.

Weinstein, J. R. and Cook, P. R. 1997. FAUST: A Framework for Algorithm Understanding and Sonification Testing, *Proceedings of the International Conference on Auditory Display (ICAD'97)*, Palo Alto, California, USA, pp. 97–104.

Weiss, R. 1996. DSPs Wrestle With CPUs in the Embedded Arena. *Computer Design*, 35(4), pp. 75–87.

Wright, M. and Freed, A. 1997. Open SoundControl: A New Protocol for Communicating with Sound Synthesizers, *Proceedings of the 1997 International Computer Music Conference (ICMC'97)*, Thessaloniki, Greece, pp. 101–104.

Zetterberg, L. H. and Zhang Q. 1988. Elimination of Transients in Adaptive Filters with Application to Speech Coding. *Signal Processing*, 15(4), pp. 419–428.