

DAFX: Digital Audio Effects

Udo Zolzer

Copyright © 2002 John Wiley & Sons, Ltd

ISBNs: 0-471-49078-4 (Hardback); 0-470-84604-6 (Electronic)

DAFX - Digital Audio Effects

DAFX - Digital Audio Effects

Udo Zölzer, Editor

University of the Federal Armed Forces, Hamburg, Germany

Xavier Amatriain

Pompeu Fabra University, Barcelona, Spain

Daniel Arfib

CNRS - Laboratoire de Mécanique et d'Acoustique, Marseille, France

Jordi Bonada

Pompeu Fabra University, Barcelona, Spain

Giovanni De Poli

University of Padova, Italy

Pierre Dutilleux

Lübeck, Germany

Gianpaolo Evangelista

Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland

Florian Keiler

University of the Federal Armed Forces, Hamburg, Germany

Alex Loscos

Pompeu Fabra University, Barcelona, Spain

Davide Rocchesso

University of Verona, Italy

Mark Sandler

Queen Mary, University of London, United Kingdom

Xavier Serra

Pompeu Fabra University, Barcelona, Spain

Todor Todoroff

Brussels, Belgium



JOHN WILEY & SONS, LTD

Copyright ©2002 by John Wiley & Sons, Ltd
Baffins Lane, Chichester,
West Sussex, PO 19 1UD, England

National 01243 779777
International (+44) 1243 779777
e-mail (for orders and customer service enquiries): cs-books@wiley.co.uk

Visit our Home Page on <http://www.wiley.co.uk>
or <http://www.wiley.com>

All Rights Reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except under the terms of the Copyright Designs and Patents Act 1988 or under the terms of a licence issued by the Copyright Licensing Agency, 90 Tottenham Court Road, London, W1P 0LP, UK, without the permission in writing of the Publisher, with the exception of any material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the publication.

Neither the authors nor John Wiley & Sons, Ltd accept any responsibility or liability for loss or damage occasioned to any person or property through using the material, instructions, methods or ideas contained herein, or acting or refraining from acting as a result of such use. The authors and Publisher expressly disclaim all implied warranties, including merchantability of fitness for any particular purpose. There will be no duty on the authors or Publisher to correct any errors or defects in the software.

Designations used by companies to distinguish their products are often claimed as trademarks. In all instances where John Wiley & Sons, Ltd is aware of a claim, the product names appear in initial capital or capital letters. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

Other Wiley Editorial Offices

John Wiley & Sons, Inc., 605 Third Avenue,
New York, NY 10158-0012, USA

WILEY-VCH Verlag GmbH
Pappelallee 3, D-69469 Weinheim, Germany

John Wiley & Sons Australia, Ltd, 33 Park Road, Milton,
Queensland 4064, Australia

John Wiley & Sons (Canada) Ltd, 22 Worcester Road
Rexdale, Ontario, M9W 1L1, Canada

John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop #02-01,
Jin Xing Distripark, Singapore 129809

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

ISBN 0 471 49078 4

Produced from PostScript files supplied by the author.

Printed and bound in Great Britain by Biddles Ltd, Guildford and King's Lynn.

This book is printed on acid-free paper responsibly manufactured from sustainable forestry,
in which at least two trees are planted for each one used for paper production.

Contents

Preface	xiv
List of Contributors	xvi
1 Introduction	1
U. Zölzer	
1.1 Digital Audio Effects DAFX with MATLAB	1
1.2 Fundamentals of Digital Signal Processing	2
1.2.1 Digital Signals	3
1.2.2 Spectrum Analysis of Digital Signals	6
1.2.3 Digital Systems	18
1.3 Conclusion	29
Bibliography	29
2 Filters	31
<i>P. Dutilleux, U. Zölzer</i>	
2.1 Introduction	31
2.2 Basic Filters	33
2.2.1 Lowpass Filter Topologies	33
2.2.2 Parametric AP, LP, HP, BP and BR Filters	38
2.2.3 FIR Filters	45
2.2.4 Convolution	48
2.3 Equalizers	50
2.3.1 Shelving Filters	51
2.3.2 Peak Filters	52
2.4 Time-varying Filters	55

2.4.1	Wah-wah Filter	55
2.4.2	Phaser	56
2.4.3	Time-varying Equalizers	58
2.5	Conclusion	59
	Sound and Music	59
	Bibliography	60
3	Delays	63
<i>P. Dutilleux, U. Zölzer</i>		
3.1	Introduction	63
3.2	Basic Delay Structures	63
3.2.1	FIR Comb Filter	63
3.2.2	IIR Comb Filter	64
3.2.3	Universal Comb Filter	65
3.2.4	Fractional Delay Lines	66
3.3	Delay-based Audio Effects	68
3.3.1	Vibrato	68
3.3.2	Flanger, Chorus, Slapback, Echo	69
3.3.3	Multiband Effects	71
3.3.4	Natural Sounding Comb Filter	72
3.4	Conclusion	73
	Sound and Music	73
	Bibliography	73
4	Modulators and Demodulators	75
<i>P. Dutilleux, U. Zölzer</i>		
4.1	Introduction	75
4.2	Modulators	76
4.2.1	Ring Modulator	76
4.2.2	Amplitude Modulator	77
4.2.3	Single-Side Band Modulator	77
4.2.4	Frequency and Phase Modulator	80
4.3	Demodulators	82
4.3.1	Detectors	83
4.3.2	Averagers	83
4.3.3	Amplitude Scalers	84

<i>Contents</i>	vii
-----------------	-----

4.3.4 Typical Applications	84
4.4 Applications	85
4.4.1 Vibrato	86
4.4.2 Stereo Phaser	86
4.4.3 Rotary Loudspeaker Effect	86
4.4.4 SSB Effects	88
4.4.5 Simple Morphing: Amplitude Following	88
4.5 Conclusion	90
Sound and Music	91
Bibliography	91
5 Nonlinear Processing	93
<i>P. Dutilleux, U. Zölzer</i>	
5.1 Introduction	93
5.2 Dynamics Processing	95
5.2.1 Limiter	99
5.2.2 Compressor and Expander	100
5.2.3 Noise Gate	102
5.2.4 De-esser	104
5.2.5 Infinite Limiters	105
5.3 Nonlinear Processors	106
5.3.1 Basics of Nonlinear Modeling	106
5.3.2 Valve Simulation	109
5.3.3 Overdrive, Distortion and Fuzz	116
5.3.4 Harmonic and Subharmonic Generation	126
5.3.5 Tape Saturation	128
5.4 Exciters and Enhancers	128
5.4.1 Exciters	128
5.4.2 Enhancers	131
5.5 Conclusion	132
Sound and Music	133
Bibliography	133

6 Spatial Effects	137
<i>D. Rocchesso</i>	
6.1 Introduction	137
6.2 Basic Effects	138
6.2.1 Panorama	138
6.2.2 Precedence Effect	141
6.2.3 Distance and Space Rendering	143
6.2.4 Doppler Effect	145
6.2.5 Sound Trajectories	147
6.3 3D with Headphones	149
6.3.1 Localization	149
6.3.2 Interaural Differences	151
6.3.3 Externalization	151
6.3.4 Head-Related Transfer Functions	154
6.4 3D with Loudspeakers	159
6.4.1 Introduction	159
6.4.2 Localization with Multiple Speakers	160
6.4.3 3D Panning	161
6.4.4 Ambisonics and Holophony	163
6.4.5 Transaural Stereo	165
6.4.6 Room-Within-the-Room Model	167
6.5 Reverberation	170
6.5.1 Acoustic and Perceptual Foundations	170
6.5.2 Classic Reverberation Tools	177
6.5.3 Feedback Delay Networks	180
6.5.4 Convolution with Room Impulse Responses	184
6.6 Spatial Enhancements	186
6.6.1 Stereo Enhancement	186
6.6.2 Sound Radiation Simulation	191
6.7 Conclusion	193
Sound and Music	193
Bibliography	194

7 Time-segment Processing	201
<i>P. Dutilleux, G. De Poli, U. Zölzer</i>	
7.1 Introduction	201
7.2 Variable Speed Replay	202
7.3 Time Stretching	205
7.3.1 Historical Methods - Phonogène	207
7.3.2 Synchronous Overlap and Add (SOLA)	208
7.3.3 Pitch-synchronous Overlap and Add (PSOLA)	211
7.4 Pitch Shifting	215
7.4.1 Historical Methods - Harmonizer	216
7.4.2 Pitch Shifting by Time Stretching and Resampling	217
7.4.3 Pitch Shifting by Delay Line Modulation	220
7.4.4 Pitch Shifting by PSOLA and Formant Preservation	222
7.5 Time Shuffling and Granulation	226
7.5.1 Time Shuffling	226
7.5.2 Granulation	229
7.6 Conclusion	232
Sound and Music	233
Bibliography	234
8 Time-frequency Processing	237
<i>D. Arfib, F. Keiler, U. Zölzer</i>	
8.1 Introduction	237
8.2 Phase Vocoder Basics	238
8.2.1 Filter Bank Summation Model	240
8.2.2 Block-by-Block Analysis/Synthesis Model	242
8.3 Phase Vocoder Implementations	244
8.3.1 Filter Bank Approach	246
8.3.2 Direct FFT/IFFT Approach	251
8.3.3 FFT Analysis/Sum of Sinusoids Approach	255
8.3.4 Gaboret Approach	257
8.3.5 Phase Unwrapping and Instantaneous Frequency	261
8.4 Phase Vocoder Effects	263
8.4.1 Time-frequency Filtering	263
8.4.2 Dispersion	266

8.4.3	Time Stretching	268
8.4.4	Pitch Shifting	276
8.4.5	Stable/Transient Components Separation	282
8.4.6	Mutation between Two Sounds	285
8.4.7	Robotization	287
8.4.8	Whisperization	290
8.4.9	Denoising	291
8.5	Conclusion	294
	Bibliography	295
9	Source-Filter Processing	299
	<i>D. Arfib, F. Keiler, U. Zölzer</i>	
9.1	Introduction	299
9.2	Source-Filter Separation	300
9.2.1	Channel Vocoder	301
9.2.2	Linear Predictive Coding (LPC)	303
9.2.3	Cepstrum	310
9.3	Source-Filter Transformations	315
9.3.1	Vocoding or Cross-synthesis	315
9.3.2	Formant Changing	321
9.3.3	Spectral Interpolation	328
9.3.4	Pitch Shifting with Formant Preservation	330
9.4	Feature Extraction	336
9.4.1	Pitch Extraction	337
9.4.2	Other Features	361
9.5	Conclusion	370
	Bibliography	370
10	Spectral Processing	373
	<i>X. Amatriain, J. Bonada, A. Loscos, X. Serra</i>	
10.1	Introduction	373
10.2	Spectral Models	375
10.2.1	Sinusoidal Model	376
10.2.2	Sinusoidal plus Residual Model	376
10.3	Techniques	379
10.3.1	Analysis	379

10.3.2 Feature Analysis	399
10.3.3 Synthesis	403
10.3.4 Main Analysis-Synthesis Application	409
10.4 FX and Transformations	415
10.4.1 Filtering with Arbitrary Resolution	416
10.4.2 Partial Dependent Frequency Scaling	417
10.4.3 Pitch Transposition with Timbre Preservation	418
10.4.4 Vibrato and Tremolo	420
10.4.5 Spectral Shape Shift	420
10.4.6 Gender Change	422
10.4.7 Harmonizer	423
10.4.8 Hoarseness	424
10.4.9 Morphing	424
10.5 Content-Dependent Processing	426
10.5.1 Real-time Singing Voice Conversion	426
10.5.2 Time Scaling	429
10.6 Conclusion	435
Bibliography	435
11 Time and Frequency Warping Musical Signals	439
<i>G. Evangelista</i>	
11.1 Introduction	439
11.2 Warping	440
11.2.1 Time Warping	440
11.2.2 Frequency Warping	441
11.2.3 Algorithms for Warping	443
11.2.4 Short-time Warping and Real-time Implementation	449
11.2.5 Time-varying Frequency Warping	453
11.3 Musical Uses of Warping	456
11.3.1 Pitch Shifting Inharmonic Sounds	456
11.3.2 Inharmonizer	458
11.3.3 Comb Filtering+Warping and Extraction of Excitation Signals in Inharmonic Sounds	459
11.3.4 Vibrato, Glissando, Trill and Flatterzunge	460
11.3.5 Morphing	460
11.4 Conclusion	462
Bibliography	462

12 Control of Digital Audio Effects	465
<i>T. Todoroff</i>	
12.1 Introduction	465
12.2 General Control Issues	466
12.3 Mapping Issues	467
12.3.1 Assignment	468
12.3.2 Scaling	469
12.4 GUI Design and Control Strategies	470
12.4.1 General GUI Issues	470
12.4.2 A Small Case Study	471
12.4.3 Specific Real-time Control Issues	472
12.4.4 GUI Mapping Issues	473
12.4.5 GUI Programming Languages	475
12.5 Algorithmic Control	476
12.5.1 Abstract Models	476
12.5.2 Physical Models	476
12.6 Control Based on Sound Features	476
12.6.1 Feature Extraction	477
12.6.2 Examples of Controlling Digital Audio Effects	478
12.7 Gestural Interfaces	478
12.7.1 MIDI Standard	479
12.7.2 Playing by Touching and Holding the Instrument	480
12.7.3 Force-feedback Interfaces	484
12.7.4 Interfaces Worn on the Body	485
12.7.5 Controllers without Physical Contact	486
12.8 Conclusion	488
Sound and Music	490
Bibliography	490
13 Bitstream Signal Processing	499
<i>M. Sandler, U. Zölzer</i>	
13.1 Introduction	499
13.2 Sigma Delta Modulation	501
13.2.1 A Simple Linearized Model of SDM	502
13.2.2 A First-order SDM System	504

<i>Contents</i>	xiii
13.2.3 Second and Higher Order SDM Systems	505
13.3 BSP Filtering Concepts	507
13.3.1 Addition and Multiplication of Bitstream Signals	508
13.3.2 SD IIR Filters	509
13.3.3 SD FIR Filters	510
13.4 Conclusion	511
Bibliography	511
Glossary	515
Bibliography	524
Index	525

Preface

DAFX is a synonym for digital audio effects. It is also the name for a European research project for co-operation and scientific transfer, namely EU-COST-G6 “Digital Audio Effects” (1997-2001). It was initiated by Daniel Arfib (CNRS, Marseille). In the past couple of years we have had four EU-sponsored international workshops/conferences on DAFX, namely, in Barcelona (DAFX-98¹), Trondheim (DAFX-99²), Verona (DAFX-00³), and Limerick (DAFX-01⁴). A variety of DAFX topics have been presented by international participants at these conferences. The papers can be found on the corresponding web sites.

This book not only reflects these conferences and workshops, it is intended as a profound collection and presentation of the main fields of digital audio effects. The contents and structure of the book were prepared by a special book work group and discussed in several workshops over the past years sponsored by the EU-COST-G6 project. However, the single chapters are the individual work of the respective authors.

Chapter 1 gives an introduction to digital signal processing and shows software implementations with the MATLAB programming tool. Chapter 2 discusses digital filters for shaping the audio spectrum and focuses on the main building blocks for this application. Chapter 3 introduces basic structures for delays and delay-based audio effects. In Chapter 4 modulators and demodulators are introduced and their applications to digital audio effects are demonstrated. The topic of nonlinear processing is the focus of Chapter 5. First, we discuss fundamentals of dynamics processing such as limiters, compressors/expanders and noise gates and then we introduce the basics of nonlinear processors for valve simulation, distortion, harmonic generators and excitors. Chapter 6 covers the wide field of spatial effects starting with basic effects, 3D for headphones and loudspeakers, reverberation and spatial enhancements. Chapter 7 deals with time-segment processing and introduces techniques for variable speed replay, time stretching, pitch shifting, shuffling and granulation. In Chapter 8 we extend the time-domain processing of Chapters 2-7. We introduce the fundamental techniques for time-frequency processing, demonstrate several implementation schemes and illustrate the variety of effects possible in the two-dimensional time-frequency domain. Chapter 9 covers the field of source-filter processing where the audio signal is modeled as a source signal and a filter. We introduce three techniques for source-filter separation and show source-filter transformations leading to audio effects such as cross-synthesis, formant changing, spectral interpolation and pitch shifting with formant preservation. The end of this chapter covers feature extraction techniques. Chapter 10 deals with spectral processing where the audio signal is represented by spectral models such as sinusoids plus a residual signal. Techniques for analysis, higher-level feature analysis and synthesis are introduced and a variety of new audio effects based on these spectral models

¹<http://www.iua.upf.es/dafx98>

²<http://www.notam.uio.no/dafx99>

³<http://profs.sci.univr.it/~dafx>

⁴<http://www.csis.ul.ie/dafx01>

are discussed. Effect applications range from pitch transposition, vibrato, spectral shape shift, gender change to harmonizer and morphing effects. Chapter 11 deals with fundamental principles of time and frequency warping techniques for deforming the time and/or the frequency axis. Applications of these techniques are presented for pitch shifting inharmonic sounds, inharmonizer, extraction of excitation signals, morphing and classical effects. Chapter 12 deals with the control of effect processors ranging from general control techniques to control based on sound features and gestural interfaces. Finally, Chapter 13 illustrates new challenges of bitstream signal representations, shows the fundamental basics and introduces filtering concepts for bitstream signal processing. MATLAB implementations in several chapters of the book illustrate software implementations of DAFX algorithms. The MATLAB files can be found on the web site <http://www.dafx.de>.

I hope the reader will enjoy the presentation of the basic principles of DAFX in this book and will be motivated to explore DAFX with the help of our software implementations. The creativity of a DAFX designer can only grow or emerge if intuition and experimentation are combined with profound knowledge of physical and musical fundamentals. The implementation of DAFX in software needs some knowledge of digital signal processing and this is where this book may serve as a source of ideas and implementation details.

Acknowledgements

I would like to thank the authors for their contributions to the chapters and also the EU-Cost-G6 delegates from all over Europe for their contributions during several meetings and especially Nicola Bernadini, Javier Casajús, Markus Erne, Mikael Fernström, Eric Feremans, Emmanuel Favreau, Alois Melka, Jørn Rudi, and Jan Tro. The book cover is based on a mapping of a time-frequency representation of a musical piece onto the globe by Jørn Rudi. Jørn has also published a CD-ROM⁵ for making computer music “DSP-Digital Sound Processing”, which may serve as a good introduction to sound processing and DAFX. Thanks to Catja Schümann for her assistance in preparing drawings and L^AT_EX formatting, Christopher Duxbury for proof-reading and Vincent Verfaillie for comments and cleaning up the code lines of Chapters 8 to 10. I also express my gratitude to my staff members Udo Ahlvers, Manfred Chrobak, Florian Keiler, Harald Schorr, and Jörg Zeller at the UniBw Hamburg for providing assistance during the course of writing this book. Finally, I would like to thank Birgit Gruber, Ann-Marie Halligan, Laura Kempster, Susan Dunsmore, and Zoë Pinnock from John Wiley & Sons, Ltd for their patience and assistance.

My special thanks are directed to my wife Elke and our daughter Franziska.

Hamburg, March 2002

Udo Zölzer

⁵<http://www.notam.no>

List of Contributors

Xavier Amatriain was born in Barcelona in 1973. He studied Telecommunications Engineering at the UPC (Barcelona) where he graduated in 1999. In the same year he joined the Music Technology Group in the Audiovisual Institute (Pompeu Fabra University). He is currently a lecturer at the same university where he teaches Software Engineering and Audio Signal Processing and is also a PhD candidate. His past research activities include participation in the MPEG-7 development task force as well as projects dealing with synthesis control and audio analysis. He is currently involved in research in the fields of spectral analysis and the development of new schemes for content-based synthesis and transformations.

Daniel Arfib (born 1949) received his diploma as “ingénieur ECP” from the Ecole Centrale of Paris in 1971 and is a “docteur-ingénieur” (1977) and “docteur es sciences” (1983) from the Université of Marseille II. After a few years in education or industry jobs, he has devoted his work to research, joining the CNRS (National Center for Scientific Research) in 1978 at the Laboratory of Mechanics and Acoustics (LMA) of Marseille (France). His main concern is to provide a combination of scientific and musical points on views on synthesis, transformation and interpretation of sounds using the computer as a tool, both as a researcher and a composer. As the chairman of the COST-G6 action named “Digital Audio Effects” he has been in the middle of a galaxy of researchers working on this subject. He also has a strong interest in the gesture and sound relationship, especially concerning creativity in musical systems.

Jordi Bonada studied Telecommunication Engineering at the Catalunya Polytechnic University of Barcelona (Spain) and graduated in 1997. In 1996 he joined the Music Technology Group of the Audiovisual Institute of the UPF as a researcher and developer in digital audio analysis and synthesis. Since 1999 he has been a lecturer at the same university where he teaches Audio Signal Processing and is also a PhD candidate in Informatics and Digital Communication. He is currently involved in research in the fields of spectral signal processing, especially in audio time-scaling and voice synthesis and modeling.

Giovanni De Poli is an Associate Professor of Computer Science at the Department of Electronics and Informatics of the University of Padua, where he teaches “Data Structures and Algorithms” and “Processing Systems for Music”. He is the Director of the Centro di Sonologia Computazionale (CSC) of the University of Padua. He is a member of the Executive Committee (ExCom) of the IEEE Computer Society Technical Committee on Computer Generated Music, member of the Board of Directors of AIMI (Associazione Italiana di Informatica Musicale), member of the Board of Directors of CIARM (Centro Interuniversitario di Acustica e Ricerca Musicale), member of the Scientific Committee of ACROE (Institut National Polytechnique Grenoble), and Associate Editor of the *International Journal of New Music Research*. His main research interests are in algorithms for sound synthesis and analysis, models for expressiveness in music, multimedia systems and human-computer interaction, and the preservation and restoration of audio documents. He is the author of several scientific international publications, and has served in

the Scientific Committees of international conferences. He is coeditor of the books *Representations of Music Signals*, MIT Press 1991, and *Musical Signal Processing*, Swets & Zeitlinger, 1996. Systems and research developed in his lab have been exploited in collaboration with digital musical instruments industry (GeneralMusic). He is the owner of patents on digital music instruments.

Pierre Dutilleux graduated in thermal engineering from the Ecole Nationale Supérieure des Techniques Industrielles et des Mines de Douai (ENSTIMD) in 1983 and in information processing from the Ecole Nationale Supérieure d'Electronique et de Radioélectricité de Grenoble (ENSERG) in 1985. He developed audio and musical applications for the Syter real-time audio processing system designed at INA-GRM by J.-F. Allouis. After developing a set of audio processing algorithms as well as implementing the first wavelet analyzer on a digital signal processor, he got a PhD in acoustics and computer music from the University of Aix-Marseille II in 1991 under the direction of J.-C. Risset. From 1991 through 2000 he worked as a research and development engineer at the ZKM (Center for Art and Media Technology) in Karlsruhe. There he planned computer and digital audio networks for a large digital audio studio complex, and he introduced live electronics and physical modeling as tools for musical production. He contributed to multimedia works with composers such as K. Furukawa and M. Maiguashca. He designed and realized the AML (Architecture and Music Laboratory) as an interactive museum installation. He is a German delegate on the Digital Audio Effects (DAFX) project. He describes himself as a “digital musical instrument builder”.

Gianpaolo Evangelista received the laurea in physics (summa cum laude) from the University of Naples, Italy, in 1984 and the MSc and PhD degrees in electrical engineering from the University of California, Irvine, in 1987 and 1990, respectively. Since 1998 he has been a Scientific Adjunct with the Laboratory for Audiovisual Communications, Swiss Federal Institute of Technology, Lausanne, Switzerland, on leave from the Department of Physical Sciences, University of Naples Federico II, which he joined in 1995 as a Research Associate. From 1985 to 1986, he worked at the Centre d'Etudes de Mathematique et Acoustique Musicale (CEMAMu/CNET), Paris, France, where he contributed to the development of a DSP-based sound synthesis system, and from 1991 to 1994, he was a Research Engineer at the Microgravity Advanced Research and Support (MARS) Center, Naples, where he was engaged in research in image processing applied to fluid motion analysis and material science. His interests include speech, music, and image processing; coding; wavelets; and multirate signal processing. Dr Evangelista was a recipient of the Fulbright fellowship.

Florian Keiler was born in Hamburg, Germany, in 1972. He studied electrical engineering at the Technical University Hamburg-Harburg. As part of the study he spent 5 months at the King's College London in 1998. There he carried out research on speech coding based on linear predictive coding (LPC). He obtained his Diplom-Ingenieur degree in 1999. He is currently working on a PhD degree at the University of the Federal Armed Forces in Hamburg. His main research field is near lossless and low-delay audio coding for a real-time implementation on a digital signal processor (DSP). He works also on musical aspects and audio effects related

to LPC and high-resolution spectral analysis.

Alex Loscos received the BSc and MSc degrees in Telecommunication Engineering from Catalunya Polytechnic University, Barcelona, Spain, in 1997 and 1999 respectively. He is currently a Ph.D. candidate in Informatics and Digital Communication at the Pompeu Fabra University (UPF) of Barcelona. In 1997 he joined the Music Technology Group of the Audiovisual Institute of the UPF as a researcher and developer. In 1999 he became a member of the Technology Department of the UPF as lecturer and he is currently teaching and doing research in voice processing/recognition, digital audio analysis/synthesis and transformations, and statistical digital signal processing and modeling.

Davide Rocchesso received the Laurea in Ingegneria Elettronica and PhD degrees from the University of Padua, Italy, in 1992 and 1996, respectively. His PhD research involved the design of structures and algorithms based on feedback delay networks for sound processing applications. In 1994 and 1995, he was a Visiting Scholar at the Center for Computer Research in Music and Acoustics (CCRMA), Stanford University, Stanford, CA. Since 1991, he has been collaborating with the Centro di Sonologia Computazionale (CSC), University of Padua as a Researcher and Live-Electronic Designer. Since 1998, he has been with the University of Verona, Italy, as an Assistant Professor. At the Dipartimento di Informatica of the University of Verona he coordinates the project “Sounding Object”, funded by the European Commission within the framework of the Disappearing Computer initiative. His main interests are in audio signal processing, physical modeling, sound reverberation and spatialization, multimedia systems, and human-computer interaction.

Mark Sandler (born 1955) is Professor of Signal Processing at Queen Mary, University of London, where he moved in 2001 after 19 years at King's College London. He was founder and CEO of Insonify Ltd, an Internet Streaming Audio start-up for 18 months. Mark received the BSc and PhD degrees from University of Essex, UK, in 1978 and 1984, respectively. He has published over 200 papers in journals and conferences. He is a Senior Member of IEEE, a Fellow of IEE and a Fellow of the Audio Engineering Society. He has worked in Digital Audio for over 20 years on a wide variety of topics including: Digital Power amplification; Drum Synthesis; Chaos and Fractals for Analysis and Synthesis; Digital EQ; Wavelet Audio Codecs; Sigma-Delta Modulation and Direct Stream Digital technologies; Broadcast Quality Speech Coding; Internet Audio Streaming; automatic music transcription, 3D sound reproduction; processing in the compression domain, high quality audio compression, non-linear dynamics, and time stretching. Living in London, he has a wife, Valerie, and 3 children, Rachel, Julian and Abigail, aged 9, 7 and 5 respectively. A great deal of his spare time is happily taken up playing with the children or playing cricket.

Xavier Serra (born in 1959) is the Director of the Audiovisual Institute (IUA) and the head of the Music Technology Group at the Pompeu Fabra University (UPF) in Barcelona, where he has been Associate Professor since 1994. He holds a Master degree in Music from Florida State University (1983), a PhD in Computer Music from Stanford University (1989) and has worked as Chief Engineer in Yamaha Music Technologies USA, Inc. His research interests are in sound analysis and synthesis for

music and other multimedia applications. Specifically, he is working with spectral models and their application to synthesis, processing, high quality coding, plus other music-related problems such as: sound source separation, performance analysis and content-based retrieval of audio.

Todor Todoroff (born in 1963), is an electrical engineer with a specialization in telecommunications. He received a First Prize in Electroacoustic Composition at the Royal Conservatory of Music in Brussels as well as a higher diploma in Electroacoustic Composition at the Royal Conservatory of Music in Mons in the class of Annette Vande Gorne. After having been a researcher in the field of speech processing at the Free University of Brussels, for 5 years he was head of the Computer Music Research at the Polytechnic Faculty in Mons (Belgium) where he developed real-time software tools for processing and spatialization of sounds aimed at electroacoustic music composers in collaboration with the Royal Conservatory of Music in Mons. He collaborated on many occasions with IRCAM where his computer tools were used by composers Emmanuel Nunes, Luca Francesconi and Joshua Fineberg. His programs were used in Mons by composers like Leo Kupper, Robert Normandieu and Annette Vande Gorne. He continues his research within ARTeM where he developed a sound spatialization audio matrix and interactive systems for sound installations and dance performances. He is co-founder and president of ARTeM (Art, Research, Technology & Music) and FeBeME (Belgian Federation of Electroacoustic Music), administrator of NICE and member of the Bureau of ICEM. He is a Belgian representative of the European COST-G6 Action “Digital Audio Effects”. His electroacoustic music shows a special interest in multiphony and sound spatialization as well as in research into new forms of sound transformation. He composes music for concert, film, video, dance, theater and sound installation.

Udo Zölzer was born in Arolsen, Germany, in 1958. He received the Diplom-Ingenieur degree in electrical engineering from the University of Paderborn in 1985, the Dr.-Ingenieur degree from the Technical University Hamburg-Harburg (TUHH) in 1989 and completed a *habilitation* in Communications Engineering at the TUHH in 1997. Since 1999 he has been a Professor and head of the Department of Signal Processing and Communications at the University of the Federal Armed Forces in Hamburg, Germany. His research interests are audio and video signal processing and communication. He has worked as a consultant for several companies in related fields. He is a member of the AES and the IEEE. In his free time he enjoys listening to music and playing the guitar and piano.

Chapter 1

Introduction

U. Zölzer

1.1 Digital Audio Effects DAFX with MATLAB

Audio effects are used by all individuals involved in the generation of musical signals and start with special playing techniques by musicians, merge to the use of special microphone techniques and migrate to effect processors for synthesizing, recording, production and broadcasting of musical signals. This book will cover several categories of sound or audio effects and their impact on sound modifications. Digital audio effects - as an acronym we use DAFX - are boxes or software tools with input audio signals or sounds which are modified according to some sound control parameters and deliver output signals or sounds (see Fig. 1.1). The input and output signals are monitored by loudspeakers or headphones and some kind of visual representation of the signal such as the time signal, the signal level and its spectrum. According to acoustical criteria the sound engineer or musician sets his control parameters for the sound effect he would like to achieve. Both input and output

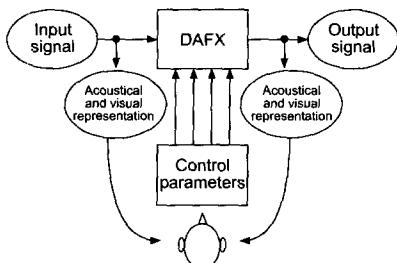


Figure 1.1 Digital audio effect and its control [Arf99].

signals are in digital format and represent analog audio signals. Modification of the sound characteristic of the input signal is the main goal of digital audio effects. The settings of the control parameters are often done by sound engineers, musicians or simply the music listener, but can also be part of the digital audio effect.

The aim of this book is the description of digital audio effects with regard to

- *physical and acoustical effect*: we take a short look at the physical background and explanation. We describe analog means or devices which generate the sound effect.
- *digital signal processing*: we give a formal description of the underlying algorithm and show some implementation examples.
- *musical applications*: we point out some applications and give references to sound examples available on CD or on the WEB.

The physical and acoustical phenomena of digital audio effects will be presented at the beginning of each effect description, followed by an explanation of the signal processing techniques to achieve the effect and some musical applications and the control of effect parameters.

In this introductory chapter we next explain some simple basics of digital signal processing and then show how to write simulation software for audio effects processing with the MATLAB¹ simulation tool or freeware simulation tools². MATLAB implementations of digital audio effects are a long way from running in real-time on a personal computer or allowing real-time control of its parameters. Nevertheless the programming of signal processing algorithms and in particular sound effect algorithms with MATLAB is very easy and can be learned very quickly.

1.2 Fundamentals of Digital Signal Processing

The fundamentals of digital signal processing consist of the description of *digital signals* - in the context of this book we use digital audio signals - as a sequence of numbers with appropriate number representation and the description of *digital systems*, which are described by software algorithms to calculate an output sequence of numbers from an input sequence of numbers. The visual representation of digital systems is achieved by functional block diagram representation or signal flow graphs. We will focus on some simple basics as an introduction to the notation and refer the reader to the literature for an introduction to digital signal processing [ME93, Orf96, Zöl97, MSY98, Mit01].

¹<http://www.mathworks.com>

²<http://www.octave.org>

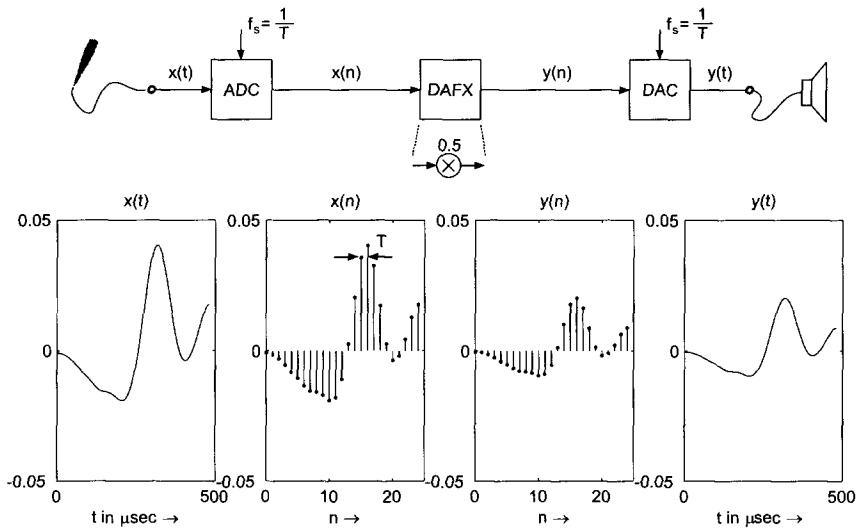


Figure 1.2 Sampling and quantizing by ADC, digital audio effects and reconstruction by DAC.

1.2.1 Digital Signals

The digital signal representation of an analog audio signal as a sequence of numbers is achieved by an analog-to-digital converter ADC. The ADC performs *sampling* of the amplitudes of the analog signal $x(t)$ on an equidistant grid along the horizontal time axis and *quantization* of the amplitudes to fixed samples represented by numbers $x(n)$ along the vertical amplitude axis (see Fig. 1.2). The samples are shown as vertical lines with dots on the top. The analog signal $x(t)$ denotes the signal amplitude over continuous time t in μ sec. Following the ADC, the digital (discrete-time and quantized amplitude) signal is represented by a sequence (stream) of samples $x(n)$ represented by numbers over the discrete time index n . The time distance between two consecutive samples is termed *sampling interval* T (sampling period) and the reciprocal is the *sampling frequency* $f_s = 1/T$ (sampling rate). The sampling frequency reflects the number of samples per second in Hertz (Hz). According to the sampling theorem, it has to be chosen as twice the highest frequency f_{\max} (signal bandwidth) contained in the analog signal, namely $f_s > 2 \cdot f_{\max}$. If we are forced to use a fixed sampling frequency f_s , we have to make sure that our input signal we want to sample has a bandwidth according to $f_{\max} = f_s/2$. If not, we have to reject higher frequencies by filtering with a lowpass filter which passes all frequencies up to f_{\max} . The digital signal is then passed to a DAFX box (digital system), which performs a simple multiplication of each sample by 0.5 to deliver the output signal $y(n) = 0.5 \cdot x(n)$. This signal $y(n)$ is then forwarded to a digital-to-analog converter DAC, which reconstructs the analog signal $y(t)$. The output signal $y(t)$ has half the amplitude of the input signal $x(t)$. The following M-file 1.1 may serve as an example for plotting digital signals as shown in Fig. 1.2.

M-file 1.1 (figure1_02.m)

```
subplot(2,4,1);
plot((0:96)*5,y(1:97));
title('x(t)');
axis([0 500 -0.05 0.05]);
xlabel('t in \musec \rightarrow');
subplot(2,4,2);
stem((0:24),u1(1:25),'.');axis([0 25 -0.05 0.05]);
xlabel('n \rightarrow');
title('x(n)');
subplot(2,4,3);
stem((0:24),0.5*u1(1:25),'.');axis([0 25 -0.05 0.05]);
xlabel('n \rightarrow');
title('y(n)');
subplot(2,4,4);
plot((0:96)*5,0.5*y(1:97));axis([0 500 -0.05 0.05]);
xlabel('t in \mu sec \rightarrow');
title('y(t)');
```

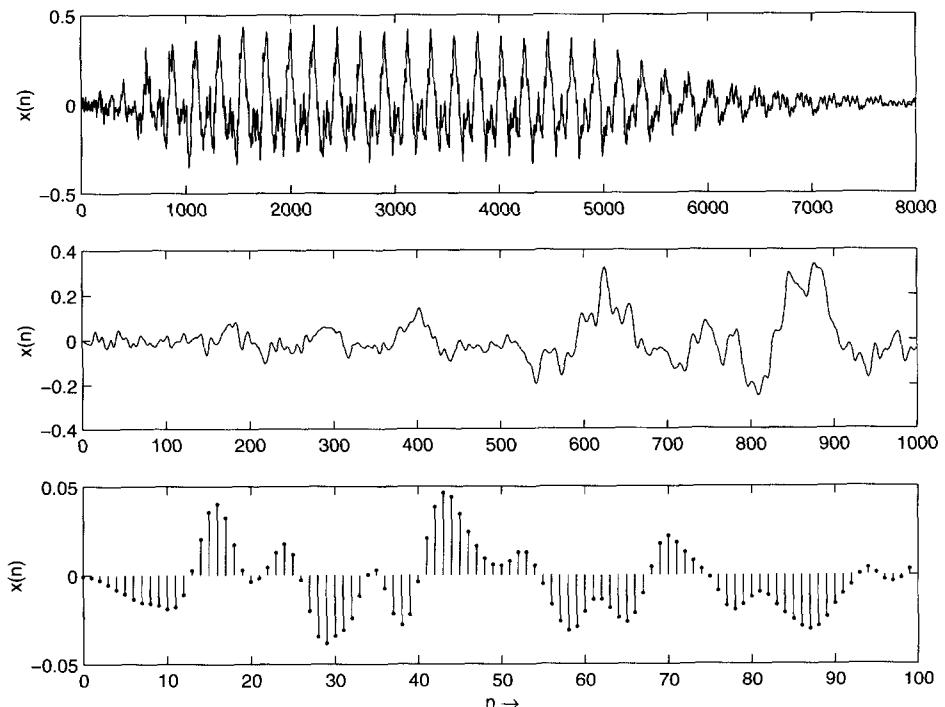


Figure 1.3 Different time representations for digital audio signals.

Figure 1.3 shows some digital signals to demonstrate different graphical representations (see M-file 1.2). The upper part shows 8000 samples, the middle part the first 1000 samples and the lower part shows the first 100 samples out of a digital audio signal. Only if the number of samples inside a figure is sufficiently low, will the line with dot graphical representation be used for a digital signal.

M-file 1.2 (figure1_03.m)

```
[u1,FS,NBITS]=wavread('ton2.wav');
```

```
figure(1)
subplot(3,1,1);
plot(0:7999,u1(1:8000));ylabel('x(n)');
subplot(3,1,2);
plot(0:999,u1(1:1000));ylabel('x(n)');
subplot(3,1,3);
stem(0:99,u1(1:100),'.');ylabel('x(n)');
xlabel('n \rightarrow');
```

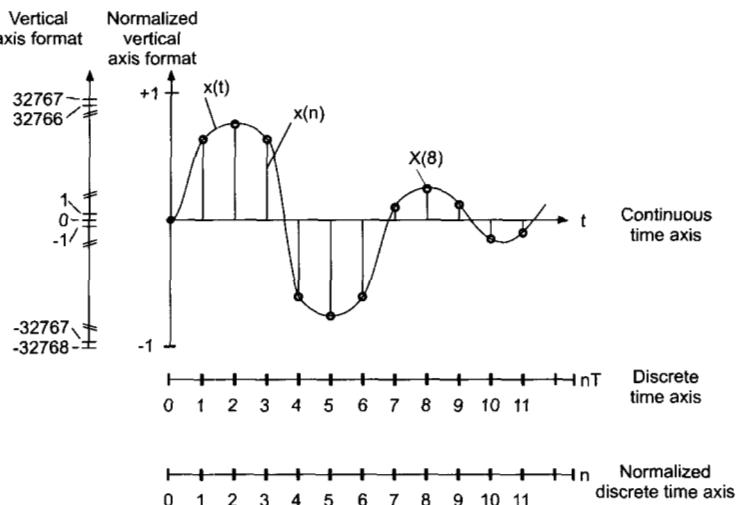


Figure 1.4 Vertical and horizontal scale formats for digital audio signals.

Two different vertical scale formats for digital audio signals are shown in Fig. 1.4. The quantization of the amplitudes to fixed numbers in the range between $-32768 \dots 32767$ is based on a 16-bit representation of the sample amplitudes which allows 2^{16} quantized values in the range $-2^{15} \dots 2^{15} - 1$. For a general w -bit representation the number range is $-2^{w-1} \dots 2^{w-1} - 1$. This representation is called the integer number representation. If we divide all integer numbers by the maximum absolute value, for example 32768, we come to the normalized vertical scale in Fig. 1.4 which is in the range between $-1 \dots 1-Q$. Q is the quantization step size and can be calculated by $Q = 2^{-(w-1)}$, which leads to $Q = 3.0518e-005$ for $w = 16$. Figure

1.4 also displays the horizontal scale formats, namely the continuous-time axis, the discrete-time axis and the normalized discrete-time axis, which will be used normally. After this narrow description we can define a digital signal as a discrete-time and discrete-amplitude signal, which is formed by sampling an analog signal and by quantization of the amplitude onto a fixed number of amplitude values. The digital signal is represented by a sequence of numbers $x(n)$. Reconstruction of analog signals can be performed by DACs. Further details of ADCs and DACs and the related theory can be found in the literature. For our discussion of digital audio effects this short introduction to digital signals is sufficient.

Signal processing algorithms usually process signals by either *block processing* or *sample-by-sample processing*. Examples for digital audio effects are presented in [Arf98]. For block processing, data are transferred to a memory buffer and then processed each time the buffer is filled with new data. Examples of such algorithms are *fast Fourier transforms (FFTs)* for spectra computations and *fast convolution*. In sample processing algorithms, each input sample is processed on a sample-by-sample basis.

A basic algorithm for weighting of a sound $x(n)$ (see Fig. 1.2) by a constant factor a demonstrates a sample-by-sample processing. (see M-file 1.3). The input signal is represented by a vector of numbers $x(0), x(1), \dots, x(\text{length}(x) - 1)$.

M-file 1.3 (sbs_alg.m)

```
% Read input sound file into vector x(n) and sampling frequency FS
[x,FS]=wavread('input filename');
% Sample-by sample algorithm y(n)=a*x(n)
for n=1:length(x),
    y(n)=a * x(n);
end;
% Write y(n) into output sound file with number of
% bits Nbits and sampling frequency FS
wavwrite(y,FS,Nbits,'output filename');
```

1.2.2 Spectrum Analysis of Digital Signals

The spectrum of a signal shows the distribution of energy over the frequency range. The upper part of Fig. 1.5 shows the spectrum of a short time slot of an analog audio signal. The frequencies range up to 20 kHz. The sampling and quantization of the analog signal with sampling frequency of $f_s = 40$ kHz lead to a corresponding digital signal. The spectrum of the digital signal of the same time slot is shown in the lower part of Fig. 1.5. The sampling operation leads to a replication of the baseband spectrum of the analog signal [Orf96]. The frequency contents from 0 Hz up to 20 kHz of the analog signal now also appear from 40 kHz up to 60 kHz and the folded version of it from 40 kHz down to 20 kHz. The replication of this first image of the baseband spectrum at 40 kHz will now also appear at integer multiples of the sampling frequency of $f_s = 40$ kHz. But notice that the spectrum of the digital signal from 0 up to 20 kHz shows exactly the same shape as the spectrum of the

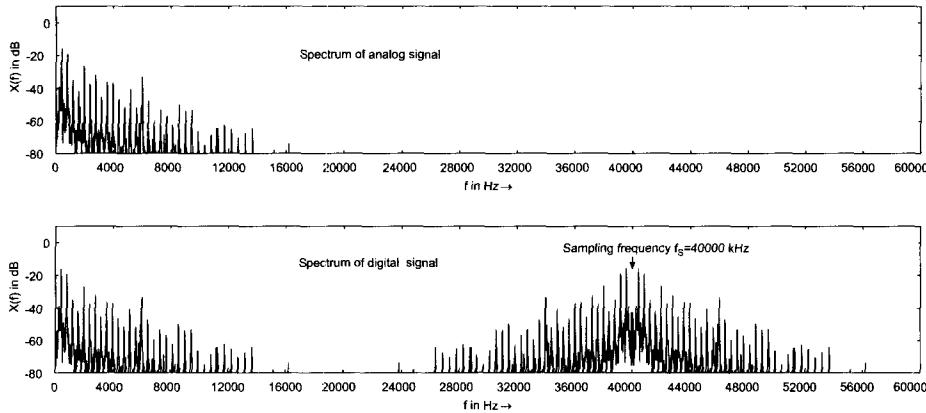


Figure 1.5 Spectra of analog and digital signals.

analog signal. The reconstruction of the analog signal out of the digital signal is achieved by simply lowpass filtering the digital signal, rejecting frequencies higher than $f_s/2 = 20$ kHz. If we consider the spectrum of the digital signal in the lower part of Fig. 1.5 and if we reject all frequencies higher than 20 kHz we come back to the spectrum of the analog signal in the upper part of the figure.

Discrete Fourier Transform

The spectrum of a digital signal can be computed by the discrete Fourier transform DFT which is given by

$$X(k) = \text{DFT}[x(n)] = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nk/N} \quad k = 0, 1, \dots, N-1. \quad (1.1)$$

The fast version of the above formula is called the fast Fourier transform FFT. The FFT takes N consecutive samples out of the signal $x(n)$ and performs a mathematical operation to yield N samples $X(k)$ of the spectrum of the signal. Figure 1.6 demonstrates the results of a 16-point FFT applied to 16 samples of a cosine signal. The result is normalized by N according to $\text{X}=\text{abs}(\text{fft}(\text{x},\text{N}))/\text{N};$.

The N samples $X(k) = X_R(k) + jX_I(k)$ are complex-valued with a real part $X_R(k)$ and an imaginary part $X_I(k)$ from which one can compute the absolute value

$$|X(k)| = \sqrt{X_R^2(k) + X_I^2(k)} \quad k = 0, 1, \dots, N-1 \quad (1.2)$$

which is the magnitude spectrum, and the phase

$$\varphi(k) = \arctan \frac{X_I(k)}{X_R(k)} \quad k = 0, 1, \dots, N-1 \quad (1.3)$$

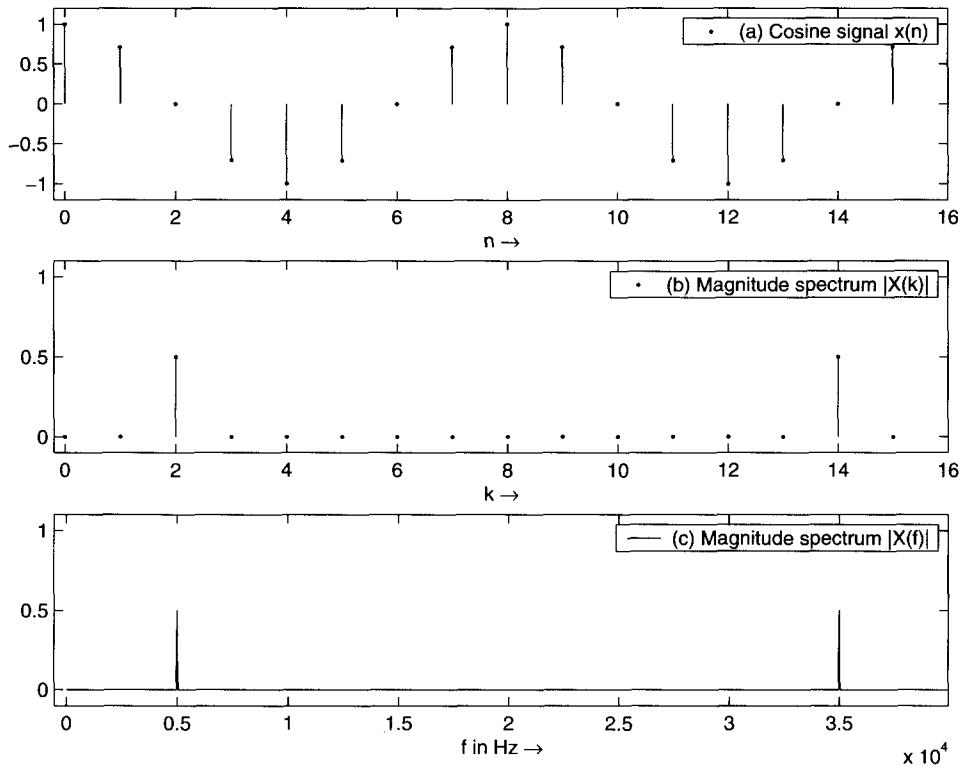


Figure 1.6 Spectrum analysis with FFT algorithm: (a) digital cosine with $N = 16$ samples, (b) magnitude spectrum $|X(k)|$ with $N = 16$ frequency samples and (c) magnitude spectrum $|X(f)|$ from 0 Hz up to the sampling frequency $f_s = 40000$ Hz.

which is the phase spectrum. Figure 1.6 also shows that the FFT algorithm leads to N equidistant frequency points which give N samples of the spectrum of the signal starting from 0 Hz in steps of $\frac{f_s}{N}$ up to $\frac{N-1}{N}f_s$. These frequency points are given by $k\frac{f_s}{N}$, where k is running from $0, 1, 2, \dots, N-1$. The magnitude spectrum $|X(f)|$ is often plotted over a logarithmic amplitude scale according to $20 \log_{10} \left(\frac{|X(f)|}{0.5} \right)$ which gives 0 dB for a sinusoid of maximum amplitude ± 1 . This normalization is equivalent to $20 \log_{10} \left(\frac{|X(k)|}{N/2} \right)$. Figure 1.7 shows this representation of the example from Fig. 1.6. Images of the baseband spectrum occur at the sampling frequency f_s and multiples of f_s . Therefore we see the original frequency at 5 kHz and in the first image spectrum the folded frequency $f_s - f_{\text{cosine}} = (40000-5)\text{Hz} = 35000$ Hz. The following M-file 1.4 is used for the computation of Figures 1.6 and 1.7.

M-file 1.4 (figure1_06_07.m)

$N=16;$

```
 $x=\cos(2*pi*2*(0:1:N-1)/N);$ 
```

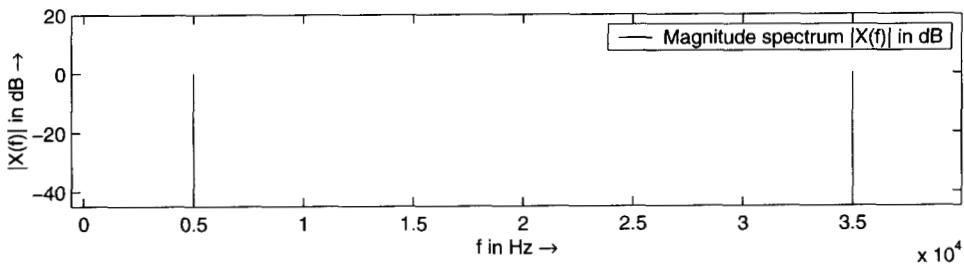


Figure 1.7 Magnitude spectrum $|X(f)|$ in dB from 0 Hz up to the sampling frequency $f_s = 40000$ Hz.

```

figure(1)
subplot(3,1,1);stem(0:N-1,x,'.');
axis([-0.2 N -1.2 1.2]);
legend('Cosine signal x(n)');
ylabel('a'));
xlabel('n \rightarrow');

X=abs(fft(x,N))/N;
subplot(3,1,2);stem(0:N-1,X,'.');
axis([-0.2 N -0.1 1.1]);
legend('Magnitude spectrum\index{Magnitude spectrum} |X(k)|');
ylabel('b'));
xlabel('k \rightarrow')

N=1024;
x=cos(2*pi*(2*1024/16)*(0:1:N-1)/N);

FS=40000;
f=((0:N-1)/N)*FS;
X=abs(fft(x,N))/N;
subplot(3,1,3);plot(f,X);
axis([-0.2*44100/16 max(f) -0.1 1.1]);
legend('Magnitude spectrum\index{Magnitude spectrum} |X(f)|');
ylabel('c'));
xlabel('f in Hz \rightarrow')

figure(2)
subplot(3,1,1);plot(f,20*log10(X./(0.5)));
axis([-0.2*44100/16 max(f) -45 20]);
legend('Magnitude spectrum\index{Magnitude spectrum} |X(f)| in dB');
ylabel('|X(f)| in dB \rightarrow');
xlabel('f in Hz \rightarrow')

```

Inverse Discrete Fourier Transform (IDFT)

Whilst the DFT is used as the transform from the discrete-time domain to the discrete-frequency domain for spectrum analysis, the inverse discrete Fourier transform IDFT allows the transform from the discrete-frequency domain to the discrete-time domain. The IDFT algorithm is given by

$$x(n) = \text{IDFT}[X(k)] = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j2\pi nk/N} \quad n = 0, 1, \dots, N-1. \quad (1.4)$$

The fast version of the IDFT is called the inverse Fast Fourier transform IFFT. Taking N complex-valued numbers and the property $X(k) = X^*(N-k)$ in the frequency domain and then performing the IFFT gives N discrete-time samples $x(n)$, which are real-valued.

Frequency Resolution: Zero-padding and Window Functions

To increase the frequency resolution for spectrum analysis we simply take more samples for the FFT algorithm. Typical numbers for the FFT resolution are $N = 256, 512, 1024, 2048, 4096$ and 8192 . If we are only interested in computing the spectrum of 64 samples and would like to increase the frequency resolution from $f_s/64$ to $f_s/1024$, we have to extend the sequence of 64 audio samples by adding zero samples up to the length 1024 and then performing an 1024-point FFT. This technique is called zero-padding and is illustrated in Fig. 1.8 and by M-file 1.5. The upper left part shows the original sequence of 8 samples and the upper right part shows the corresponding 8-point FFT result. The lower left part illustrates the adding of 8 zero samples to the original 8 sample sequence up to the length of $N = 16$. The lower right part illustrates the magnitude spectrum $|X(k)|$ resulting from the 16-point FFT of the zero-padded sequence of length $N = 16$. Notice the increase in frequency resolution between the 8-point and 16-point FFT. Between each frequency bin of the upper spectrum a new frequency bin in the lower spectrum is calculated. Bins $k = 0, 2, 4, 6, 8, 10, 12, 14$ of the 16-point FFT correspond to bins $k = 0, 1, 2, 3, 4, 5, 6, 7$ of the 8-point FFT. These N frequency bins cover the frequency range from 0 Hz up to $\frac{N-1}{N} f_s$ Hz.

```
M-file 1.5 (figure1_08.m)
x1=[-1 -0.5 1 2 2 1 0.5 -1];
x2(16)=0;
x2(1:8)=x1;

subplot(221);
stem(0:1:7,x1);axis([-0.5 7.5 -1.5 2.5]);
ylabel('x(n) \rightarrow');title('8 samples');
subplot(222);
stem(0:1:7,abs(fft(x1)));axis([-0.5 7.5 -0.5 10]);
ylabel('|X(k)| \rightarrow');title('8-point FFT');
```

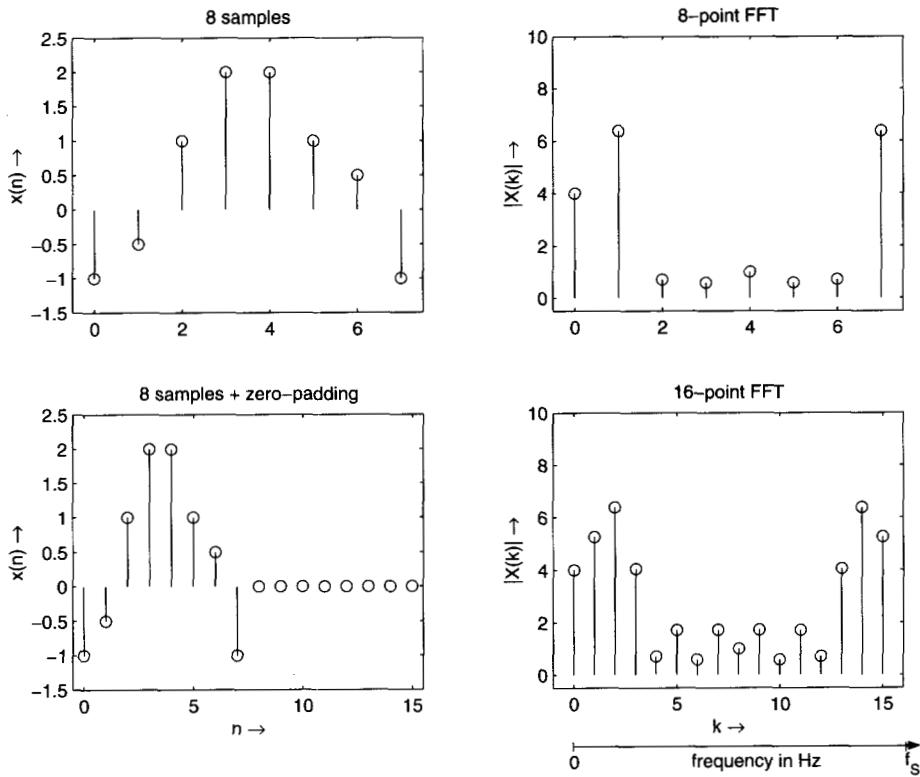


Figure 1.8 Zero-padding to increase frequency resolution.

```

subplot(223);
stem(0:1:15,x2);axis([-0.5 15.5 -1.5 2.5]);
xlabel('n \rightarrow');ylabel('x(n) \rightarrow');
title('8 samples + zero-padding');

subplot(224);
stem(0:1:15,abs(fft(x2)));axis([-1 16 -0.5 10]);
xlabel('k \rightarrow');ylabel('|X(k)| \rightarrow');
title('16-point FFT');

```

The leakage effect occurs due to cutting out N samples from the signal. This effect is shown in the upper part of Fig. 1.9 and demonstrated by the corresponding M-file 1.6. The cosine spectrum is smeared around the frequency. We can reduce the leakage effect by selecting a window function like Blackman window and

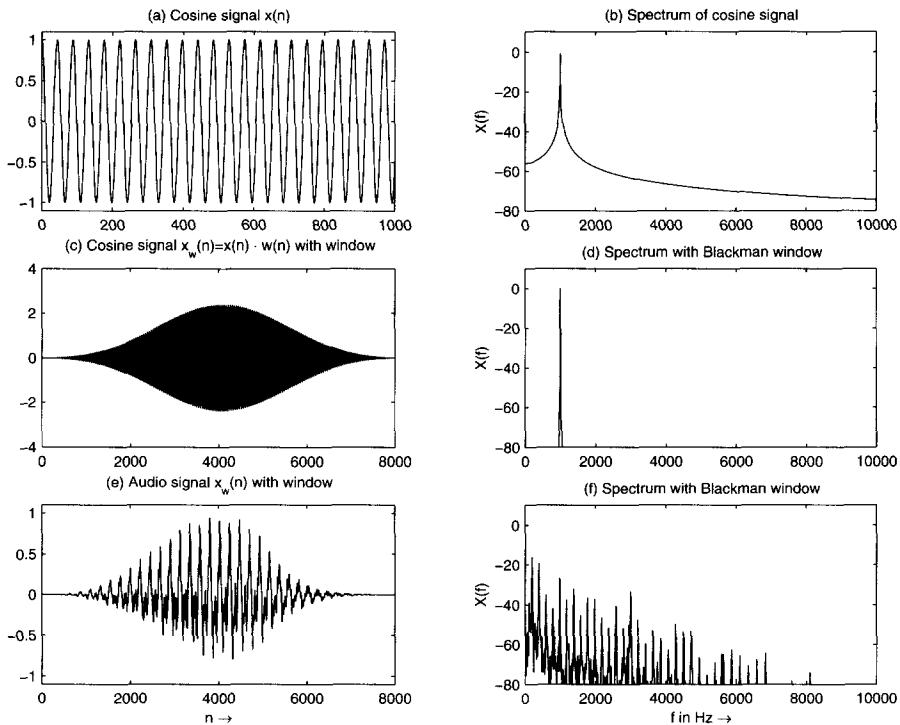


Figure 1.9 Spectrum analysis of digital signals: take N audio samples and perform an N point discrete Fourier transform to yield N samples of the spectrum of the signal starting from 0 Hz over $k \frac{f_s}{N}$ where k is running from 0, 1, 2, ..., $N-1$. (a) $x(n) = \cos(2\pi \cdot \frac{1 \text{ kHz}}{44.1 \text{ kHz}} \cdot n)$.

Hamming window

$$w_B(n) = 0.42 - 0.5 \cos(2\pi n/N) + 0.08 \cos(4\pi n/N), \quad (1.5)$$

$$w_H(n) = 0.54 - 0.46 \cos(2\pi n/N) \quad (1.6)$$

$$n = 0, 1, \dots, N-1.$$

and weighting the N audio samples by the window function. This weighting is performed according to $x_w = w(n) \cdot x(n)$ with $0 \leq n \leq N-1$ and then an FFT of the weighted signal is performed. The cosine weighted by a window and the corresponding spectrum is shown in the middle part of Fig. 1.9. The lower part of Fig. 1.9 shows a segment of an audio signal weighted by the Blackman window and the corresponding spectrum via a FFT. Figure 1.10 shows further simple examples for the reduction of the leakage effect and can be generated by the M-file 1.7.

M-file 1.6 (figure1_09.m)

```
x=cos(2*pi*1000*(0:1:N-1)/44100)';
figure(2)
```

```
W=blackman(N);
W=N*W/sum(W); % scaling of window
f=((0:N/2-1)/N)*FS;

xw=x.*W;
subplot(3,2,1);plot(0:N-1,x);
axis([0 1000 -1.1 1.1]);
title('a) Cosine signal x(n)')

subplot(3,2,3);plot(0:N-1,xw);axis([0 8000 -4 4]);
title('c) Cosine signal x_w(n)=x(n) \cdot w(n) with window')

X=20*log10(abs(fft(x,N))/(N/2));
subplot(3,2,2);plot(f,X(1:N/2));
axis([0 10000 -80 10]);
ylabel('X(f)');
title('b) Spectrum of cosine signal')

Xw=20*log10(abs(fft(xw,N))/(N/2));
subplot(3,2,4);plot(f,Xw(1:N/2));
axis([0 10000 -80 10]);
ylabel('X(f)');
title('d) Spectrum with Blackman window')

s=u1(1:N).*W;
subplot(3,2,5);plot(0:N-1,s);axis([0 8000 -1.1 1.1]);
xlabel('n \rightarrow');
title('e) Audio signal x_w(n) with window')

Sw=20*log10(abs(fft(s,N))/(N/2));
subplot(3,2,6);plot(f,Sw(1:N/2));
axis([0 10000 -80 10]);
ylabel('X(f)');
title('f) Spectrum with Blackman window')
xlabel('f in Hz \rightarrow');

M-file 1.7 (figure1_10.m)
x=[-1 -0.5 1 2 2 1 0.5 -1];
w = BLACKMAN(8);
w=w*8/sum(w);
x1=x.*w';
x2(16)=0;
x2(1:8)=x1;

subplot(421);
stem(0:1:7,x);axis([-0.5 7.5 -1.5 2.5]);
```

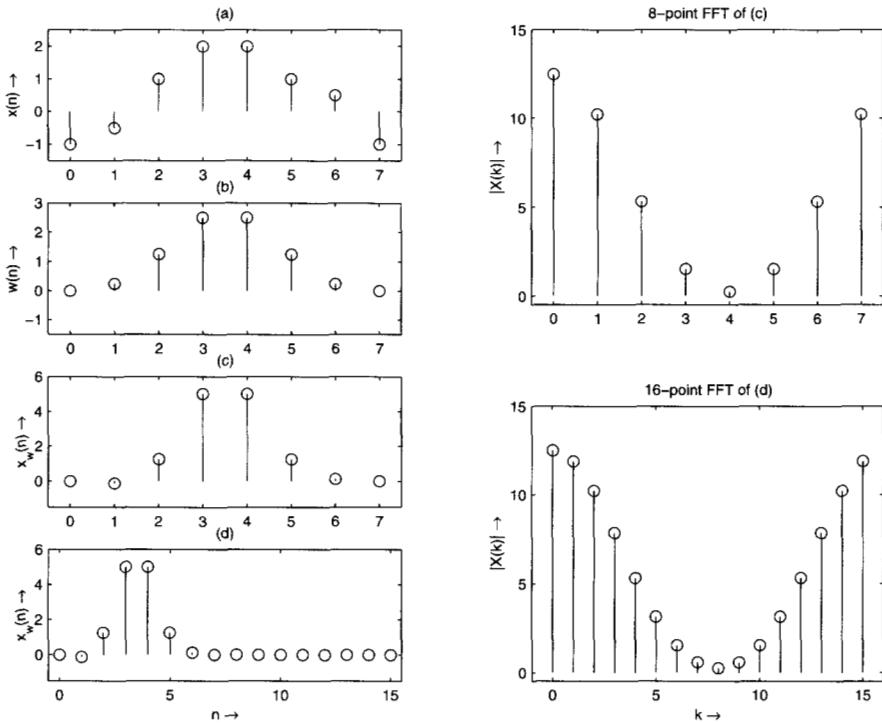


Figure 1.10 Reduction of the leakage effect by window functions: (a) the original signal, (b) the Blackman window function of length $N = 8$, (c) product $x(n) \cdot w(n)$ with $0 \leq n \leq N - 1$, (d) zero-padding applied to $x(n) \cdot w(n)$ up to length $N = 16$ and the corresponding spectra are shown on the right side.

```

ylabel('x(n) \rightarrow');
title('a) 8 samples');
subplot(423);
stem(0:1:7,w);axis([-0.5 7.5 -1.5 3]);
ylabel('w(n) \rightarrow');
title('b) 8 samples Blackman window');

subplot(425);
stem(0:1:7,x1);axis([-0.5 7.5 -1.5 6]);
ylabel('x_w(n) \rightarrow');
title('c) x(n)\cdot w(n)');

subplot(427);
stem(0:1:15,x2);axis([-0.5 15.5 -1.5 6]);
xlabel('n \rightarrow');ylabel('x_w(n) \rightarrow');
title('d) x(n)\cdot w(n) + zero-padding');

```

```

subplot(222);
stem(0:1:7,abs(fft(x1)));axis([-0.5 7.5 -0.5 15]);
ylabel('|X(k)| \rightarrow');
title('8-point FFT of c');

subplot(224);
stem(0:1:15,abs(fft(x2)));axis([-1 16 -0.5 15]);
xlabel('k \rightarrow');ylabel('|X(k)| \rightarrow');
title('16-point FFT of d');

```

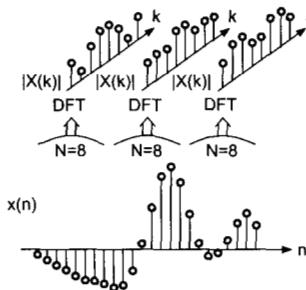


Figure 1.11 Short-time spectrum analysis by FFT.

Spectrogram: Time-frequency Representation

A special time-frequency representation is the spectrogram which gives an estimate of the short-time, time-localized frequency content of the signal. Therefore the signal is split into segments of length N which are multiplied by a window and an FFT is performed (see Fig. 1.11). To increase the time-localization of the short-time spectra an overlap of the weighted segments can be used. A special visual representation of the short-time spectra is the spectrogram in Fig. 1.12. Time increases linearly across the horizontal axis and frequency increases across the vertical axis. So each vertical line represents the absolute value $|X(f)|$ over frequency by a grey scale value (see Fig. 1.12). Only frequencies up to half the sampling frequency are shown. The calculation of the spectrogram from a signal can be performed by the MATLAB function `B = SPECGRAM(x,NFFT,Fs,WINDOW,NOVERLAP)`.

Another time-frequency representation of the short-time Fourier transforms of a signal $x(n)$ is the waterfall representation in Fig. 1.13, which can be produced by M-file 1.8 which calls the waterfall computation algorithm given by M-file 1.9.

```

M-file 1.8 (figure1_13.m)
[signal,FS,NBITS]=wavread('ton2');
subplot(211);plot(signal);
subplot(212);
waterfspec(signal,256,256,512,FS,20,-100);

```

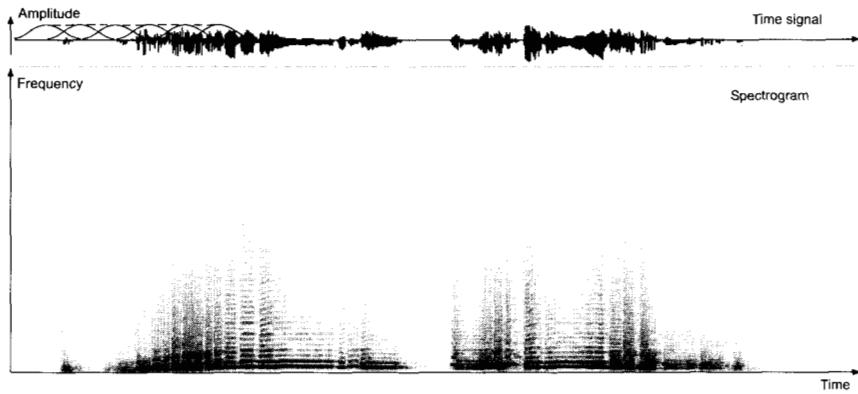


Figure 1.12 Spectrogram via FFT of weighted segments.

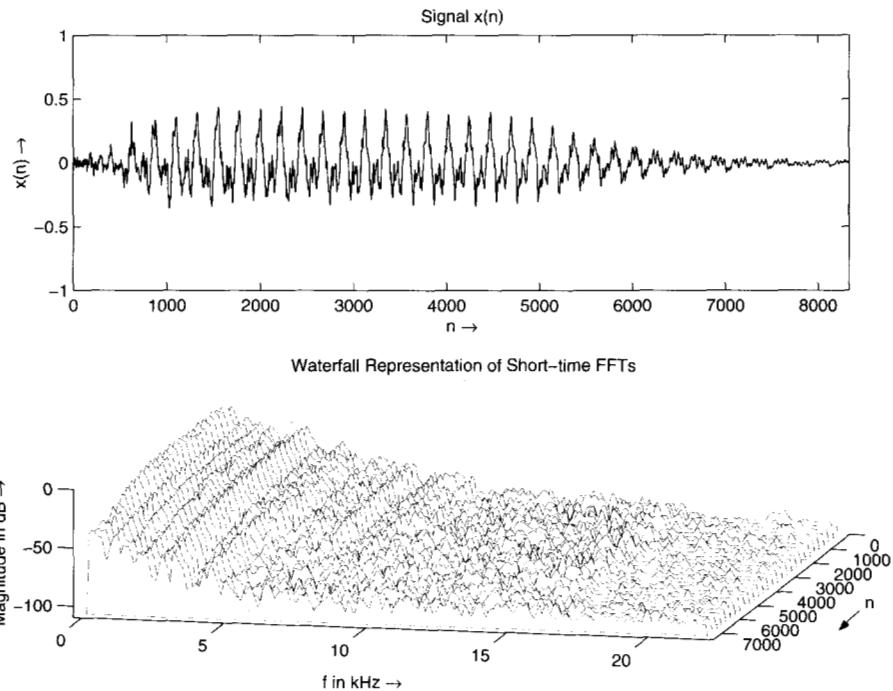


Figure 1.13 Waterfall representation via FFT of weighted segments.

M-file 1.9 (waterfspec.m)

```
function yy=waterfspec(signal,start,steps,N,fS,clippingpoint,baseplane)
% waterfspec( signal, start, steps, N, fS, clippingpoint, baseplane)
%
% shows short-time spectra of signal, starting
```

```
% at k=start, with increments of STEP with N-point FFT
% dynamic range from -baseplane in dB up to 20*log(clippingpoint)
% in dB versus time axis
%
% 18/9/98 J. Schattschneider
% 14/10/2000 U. Zoelzer
echo off;
if nargin<7, baseplane=-100; end
if nargin<6, clippingpoint=0; end
if nargin<5, fS=48000; end
if nargin<4, N=1024; end           % default FFT
if nargin<3, steps=round(length(signal)/25); end
if nargin<2, start=0; end

windoo=blackman(N);             % window - default
windoo=windoo*N/sum(windoo);     % scaling
% Calculation of number of spectra nos
n=length(signal);
rest=n-start-N;
nos=round(rest/steps);
if nos>rest/steps, nos=nos-1; end
% vectors for 3D representation
x=linspace(0, fS/1000 ,N+1);
z=x-x;
cup=z+clippingpoint;
cdown=z+baseplane;

signal=signal+0.0000001;
% Computation of spectra and visual representation
for i=1:1:nos,
    spek1=20.*log10(abs(fft(windoo.*signal(1+start+...
    ....i*steps:start+N+i*steps)))./(N)/0.5);
    spek=[-200 ; spek1(1:N)];
    spek=(spek>cup').*cup'+(spek<=cup').*spek;
    spek=(spek<cdown').*cdown'+(spek>=cdown').*spek;
    spek(1)=baseplane-10;
    spek(N/2)=baseplane-10;
    y=x-x+(i-1);
    if i==1,
        p=plot3(x(1:N/2),y(1:N/2),spek(1:N/2),'k');
        set(p,'LineWidth',0.1);
    end
    pp=patch(x(1:N/2),y(1:N/2),spek(1:N/2),'w','Visible','on');
    set(pp,'LineWidth',0.1);
end;
set(gca,'DrawMode','fast');
```

```
axis([-0.3 fS/2000+0.3 0 nos baseplane-10 0]);
set(gca,'Ydir','reverse');
view(12,40);
```

1.2.3 Digital Systems

A digital system is represented by an algorithm which uses the input signal $x(n)$ as a sequence (stream) of numbers and performs mathematical operations upon the input signal such as additions, multiplications and delay operations. The result of the algorithm is a sequence of numbers or the output signal $y(n)$. Systems which do not change their behavior over time and fulfill the superposition property [Orf96] are called linear time-invariant (LTI) systems. Nonlinear time-invariant systems will be discussed in Chapter 5. The input/output relations for a LTI digital system describe time domain relations which are based on the following terms and definitions:

- unit impulse, impulse response and discrete convolution;
- algorithms and signal flow graphs.

For each of these definitions an equivalent description in the frequency domain exists, which will be introduced later.

Unit Impulse, Impulse Response and Discrete Convolution

- Test signal: a very useful test signal for digital systems is the unit impulse

$$\delta(n) = \begin{cases} 1 & \text{for } n = 0 \\ 0 & \text{for } n \neq 0, \end{cases} \quad (1.7)$$

which is equal to one for $n = 0$ and zero elsewhere (see Fig. 1.14).

- Impulse response: if we apply a unit-sample function to a digital system, the digital system will lead to an output signal $y(n) = h(n)$, which is the so-called impulse response $h(n)$ of the digital system. The digital system is completely described by the impulse response, which is pointed out by the label $h(n)$ inside the box, as shown in Fig. 1.14.

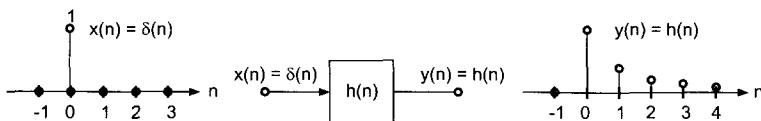


Figure 1.14 Impulse response $h(n)$ as a time domain description of a digital system.

- Discrete convolution: if we know the impulse response $h(n)$ of a digital system, we can calculate the output signal $y(n)$ from a freely chosen input signal $x(n)$ by the discrete convolution formula given by

$$y(n) = \sum_{k=-\infty}^{\infty} x(k) \cdot h(n-k) = x(n) * h(n), \quad (1.8)$$

which is often abbreviated by the second term $y(n) = x(n)*h(n)$. This discrete sum formula (1.8) represents an input-output relation for a digital system in the time domain. The computation of the convolution sum formula (1.8) can be achieved by the MATLAB function `y=conv(x,h)`.

Algorithms and Signal Flow Graphs

The above given discrete convolution formula shows the mathematical operations which have to be performed to obtain the output signal $y(n)$ for a given input signal $x(n)$. In the following we will introduce a visual representation called a signal flow graph which represents the mathematical input/output relations in a graphical block diagram. We discuss some example algorithms to show that we only need three graphical representations for the multiplication of signals by coefficients, delay and summation of signals.

- A delay of the input signal by two sampling intervals is given by the algorithm

$$y(n) = x(n-2) \quad (1.9)$$

and is represented by the block diagram in Fig. 1.15.

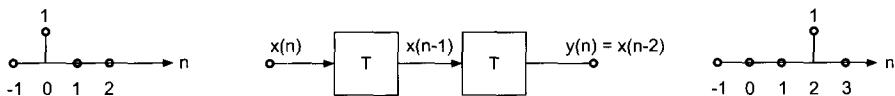


Figure 1.15 Delay of the input signal.

- A weighting of the input signal by a coefficient a is given by the algorithm

$$y(n) = a \cdot x(n) \quad (1.10)$$

and represented by a block diagram in Fig. 1.16.

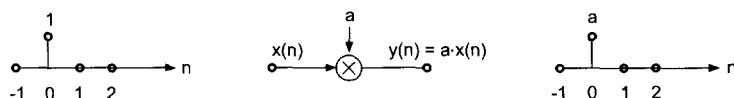


Figure 1.16 Weighting of the input signal.

- The addition of two input signals is given by the algorithm

$$y(n) = a_1 \cdot x_1(n) + a_2 \cdot x_2(n) \quad (1.11)$$

and represented by a block diagram in Fig. 1.17.

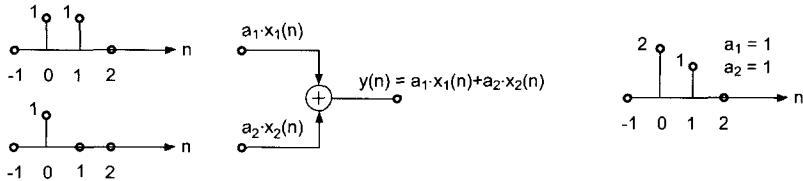


Figure 1.17 Addition of two signals $x_1(n)$ and $x_2(n)$.

- The combination of the above algorithms leads to the weighted sum over several input samples, which is given by the algorithm

$$y(n) = \frac{1}{3}x(n) + \frac{1}{3}x(n-1) + \frac{1}{3}x(n-2) \quad (1.12)$$

and represented by a block diagram in Fig. 1.18.

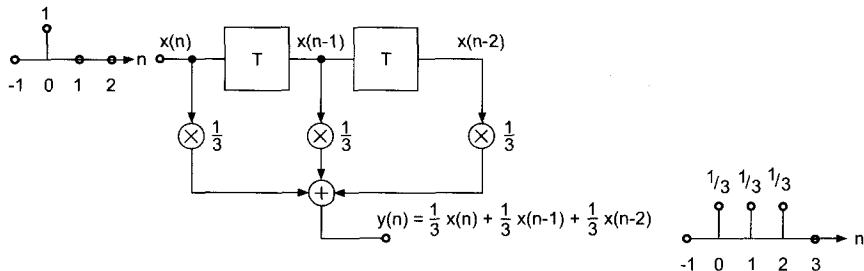


Figure 1.18 Simple digital system.

Transfer Function and Frequency Response

So far our description of digital systems has been based on the time domain relationship between the input and output signals. We noticed that the input and output signals and the impulse response of the digital system are given in the discrete time domain. In a similar way to the frequency domain description of digital signals by their spectra given in the previous subsection we can have a frequency domain description of the digital system which is represented by the impulse response $h(n)$. The frequency domain behavior of a digital system reflects its ability to pass, reject and enhance certain frequencies included in the input signal spectrum. The common terms for the frequency domain behavior are the transfer function $H(z)$ and

the frequency response $H(f)$ of the digital system. Both can be obtained by two mathematical transforms applied to the impulse response $h(n)$.

The first transform is the *Z-Transform*

$$X(z) = \sum_{n=-\infty}^{\infty} x(n) \cdot z^{-n} \quad (1.13)$$

applied to the signal $x(n)$ and the second transform is the *discrete-time Fourier transform*

$$X(e^{j\Omega}) = \sum_{n=-\infty}^{\infty} x(n) \cdot e^{-j\Omega n}, \quad (1.14)$$

$$\text{with } \Omega = \omega T = 2\pi f/f_s \quad (1.15)$$

applied to the signal $x(n)$. Both are related by the substitution $z \leftrightarrow e^{j\Omega}$. If we apply the Z-transform to the impulse response $h(n)$ of a digital system according to

$$H(z) = \sum_{n=-\infty}^{\infty} h(n) \cdot z^{-n} \quad (1.16)$$

we denote $H(z)$ as the *transfer function*. If we apply the discrete-time Fourier transform to the impulse response $h(n)$ we get

$$H(e^{j\Omega}) = \sum_{n=-\infty}^{\infty} h(n) \cdot e^{-j\Omega n}. \quad (1.17)$$

Substituting (1.15) we define the *frequency response* of the digital system by

$$H(f) = \sum_{n=-\infty}^{\infty} h(n) \cdot e^{-j2\pi f/f_s n}. \quad (1.18)$$

Causal and Stable Systems

A realizable digital system has to fulfill the following two conditions:

- Causality: a discrete-time system is *causal*, if the output signal $y(n) = 0$ for $n < 0$ for a given input signal $u(n) = 0$ for $n < 0$. This means that the system cannot react to an input before the input is applied to the system.
- Stability: a digital system is stable if

$$\sum_{n=-\infty}^{\infty} |h(n)| < M_2 < \infty \quad (1.19)$$

holds. The sum over the absolute values of $h(n)$ has to be less than a fixed number $M_2 < \infty$.

Table 1.1 Z-transforms and discrete-time Fourier transforms of $x(n)$.

Signal	Z-transform	Discrete-time Fourier transform
$x(n)$	$X(z)$	$X(e^{j\Omega})$
$x(n - M)$	$z^{-M} \cdot X(z)$	$e^{-j\Omega M} \cdot X(e^{j\Omega})$
$\delta(n)$	1	1
$\delta(n - M)$	z^{-M}	$e^{-j\Omega M}$
$x(n) \cdot e^{-j\Omega_0 n}$	$X(e^{-j\Omega_0} \cdot z)$	$X(e^{j(\Omega - \Omega_0)})$

The stability implies that the transfer function (Z-transform of impulse response) and the frequency response (discrete-time Fourier transform of impulse response) of a digital system are related by the substitution $z \leftrightarrow e^{j\Omega}$. Realizable digital systems have to be *causal* and *stable* systems. Some Z-transforms and their discrete-time Fourier transforms of a signal $x(n)$ are given in Table 1.1.

IIR and FIR Systems

IIR systems: A system with an infinite impulse response $h(n)$ is called an IIR system. From the block diagram in Fig. 1.19 we can read the difference equation

$$y(n) = x(n) - a_1 y(n-1) - a_2 y(n-2). \quad (1.20)$$

The output signal $y(n)$ is fed back through delay elements and a weighted sum of these delayed outputs is summed up to the input signal $x(n)$. Such a feedback system is also called a recursive system. The Z-transform of (1.20) yields

$$Y(z) = X(z) - a_1 z^{-1} Y(z) - a_2 z^{-2} Y(z) \quad (1.21)$$

$$X(z) = Y(z)(1 + a_1 z^{-1} + a_2 z^{-2}) \quad (1.22)$$

and solving for $Y(z)/X(z)$ gives transfer function

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{1 + a_1 z^{-1} + a_2 z^{-2}}. \quad (1.23)$$

Figure 1.20 shows a special signal flow graph representation, where adders, multipliers and delay operators are replaced by weighted graphs.

If the input delay line is extended up to $N - 1$ delay elements and the output delay line up to M delay elements according to Fig. 1.21, we can write for the difference equation

$$y(n) = - \sum_{k=1}^M a_k y(n-k) + \sum_{k=0}^{N-1} b_k x(n-k), \quad (1.24)$$

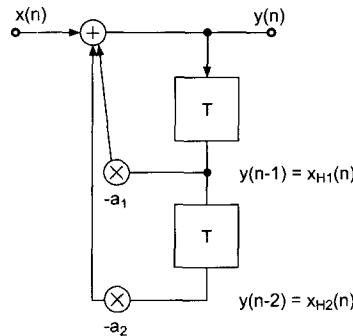


Figure 1.19 Simple IIR system with input signal $x(n)$ and output signal $y(n)$.

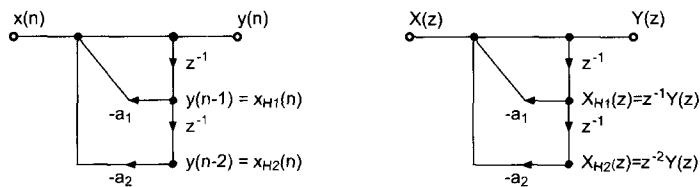


Figure 1.20 Signal flow graph of digital system in Fig. 1.19 with time domain description in the left block diagram and corresponding frequency domain description with Z-transform.

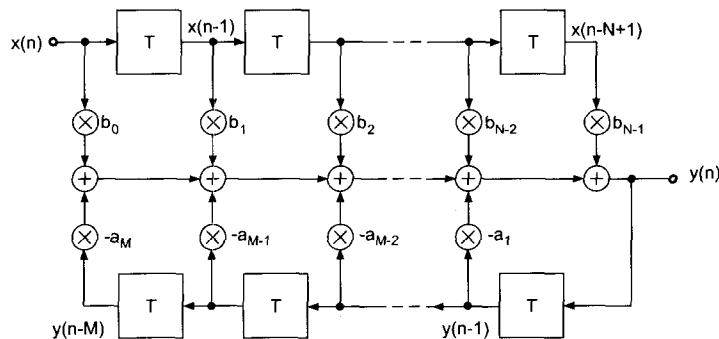


Figure 1.21 IIR system.

the Z-transform of the difference equation

$$Y(z) = - \sum_{k=1}^M a_k z^{-k} Y(z) + \sum_{k=0}^{N-1} b_k z^{-k} X(z), \quad (1.25)$$

and the resulting transfer function

$$H(z) = \frac{\sum_{k=0}^{N-1} b_k z^{-k}}{1 + \sum_{k=1}^M a_k z^{-k}}. \quad (1.26)$$

The following M-file 1.10 shows a block processing approach for the IIR filter algorithm.

M-file 1.10 (filter.m)

```
Y = FILTER(B,A,X) filters the data in vector X with the
      filter described by vectors A and B to create the filtered
      data Y. The filter is a "Direct Form II Transposed"
      implementation of the standard difference equation:
      a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(nb+1)*x(n-nb)
                           - a(2)*y(n-1) - ... - a(na+1)*y(n-na)
      If a(1) is not equal to 1, FILTER normalizes the filter
      coefficients by a(1).
```

A sample-by-sample processing approach for a second-order IIR filter algorithm is demonstrated by the following M-file 1.11.

```
M-file 1.11 (DirectForm01.m)
% M-File DirectForm01.M
% Impulse response of 2nd order IIR filter
% Sample-by-sample algorithm
clear
echo on
%
%   Impulse response of 2nd order IIR filter
%
echo off

% Coefficient computation
fg=4000;
fa=48000;
k=tan(pi*fg/fa);

b(1)=1/(1+sqrt(2)*k+k^2);
b(2)=-2/(1+sqrt(2)*k+k^2);
b(3)=1/(1+sqrt(2)*k+k^2);
a(1)=1;
a(2)=2*(k^2-1)/(1+sqrt(2)*k+k^2);
a(3)=(1-sqrt(2)*k+k^2)/(1+sqrt(2)*k+k^2);
```

```
% Initialization of state variables
xh1=0;xh2=0;
yh1=0;yh2=0;

% Input signal: unit impulse
N=20; % length of input signal
x(N)=0;x(1)=1;

% Sample-by-sample algorithm
for n=1:N
y(n)=b(1)*x(n) + b(2)*xh1 + b(3)*xh2 - a(2)*yh1 - a(3)*yh2;
xh2=xh1;xh1=x(n);
yh2=yh1;yh1=y(n);
end;

% Plot results
subplot(2,1,1)
stem(0:1:length(x)-1,x,'.');
axis([-0.6 length(x)-1 -1.2 1.2]);
xlabel('n \rightarrow');ylabel('x(n) \rightarrow');
subplot(2,1,2)
stem(0:1:length(x)-1,y,'.');
axis([-0.6 length(x)-1 -1.2 1.2]);
xlabel('n \rightarrow');ylabel('y(n) \rightarrow');
```

The computation of frequency response based on the coefficients of the transfer function $H(z) = \frac{B(z)}{A(z)}$ can be achieved by the M-file 1.12.

M-file 1.12 (freqz.m)

FREQZ Digital filter frequency response.

[H,W] = FREQZ(B,A,N) returns the N-point complex frequency response vector H and the N-point frequency vector W in radians/sample of the filter:

$$H(e) = \frac{jw}{A(e)} = \frac{jw}{B(e)} = \frac{-jw}{b(1) + b(2)e + \dots + b(m+1)e} = \frac{-jmw}{a(1) + a(2)e + \dots + a(n+1)e}$$

given numerator and denominator coefficients in vectors B and A. The frequency response is evaluated at N points equally spaced around the upper half of the unit circle.

The computation of zeros and poles of $H(z) = \frac{B(z)}{A(z)}$ is implemented by M-file 1.13.

M-file 1.13 (zplane.m)

ZPLANE Z-plane zero-pole plot.

ZPLANE(B,A) where B and A are row vectors containing transfer

function polynomial coefficients plots the poles and zeros of $B(z)/A(z)$.

FIR system: A system with a finite impulse response $h(n)$ is called an FIR system. From the block diagram in Fig. 1.22 we can read the difference equation

$$y(n) = b_0x(n) + b_1x(n - 1) + b_2x(n - 2). \quad (1.27)$$

The input signal $x(n)$ is fed forward through delay elements and a weighted sum of these delayed inputs is summed up to the input signal $y(n)$. Such a feed forward system is also called a nonrecursive system. The Z-transform of (1.27) yields

$$Y(z) = b_0X(z) + b_1z^{-1}X(z) + b_2z^{-2}X(z) \quad (1.28)$$

$$= X(z)(b_0 + b_1z^{-1} + b_2z^{-2}) \quad (1.29)$$

and solving for $Y(z)/X(z)$ gives transfer function

$$H(z) = \frac{Y(z)}{X(z)} = b_0 + b_1z^{-1} + b_2z^{-2}. \quad (1.30)$$

A general FIR system in Fig. 1.23 consists of a feed forward delay line with $N - 1$ delay elements and has the difference equation

$$y(n) = \sum_{k=0}^{N-1} b_k x(n - k). \quad (1.31)$$

The finite impulse response is given by

$$h(n) = \sum_{k=0}^{N-1} b_k \delta(n - k), \quad (1.32)$$

which shows that each impulse of $h(n)$ is represented by a weighted and shifted unit impulse. The Z-transform of the impulse response leads to the transfer function

$$H(z) = \sum_{k=0}^{N-1} b_k z^{-k}. \quad (1.33)$$

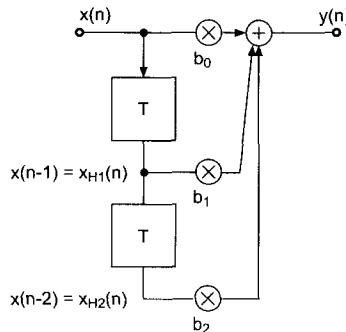


Figure 1.22 Simple FIR system with input signal $x(n)$ and output signal $y(n)$.

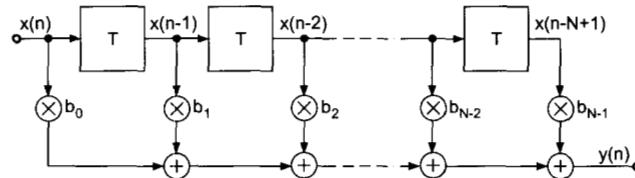


Figure 1.23 FIR system.

The time-domain algorithms for FIR systems are the same as those for IIR systems with the exception that the recursive part is missing. The previously introduced M-files for IIR systems can be used with the appropriate coefficients for FIR block processing or sample-by-sample processing.

The computation of the frequency response $H(f) = |H(f)| \cdot e^{j\angle H(f)}$ ($|H(f)|$ magnitude response, $\varphi = \angle H(f)$ phase response) from the Z-transform of an FIR impulse response according to (1.33) is shown in Fig. 1.24 and is calculated by the following M-file 1.14.

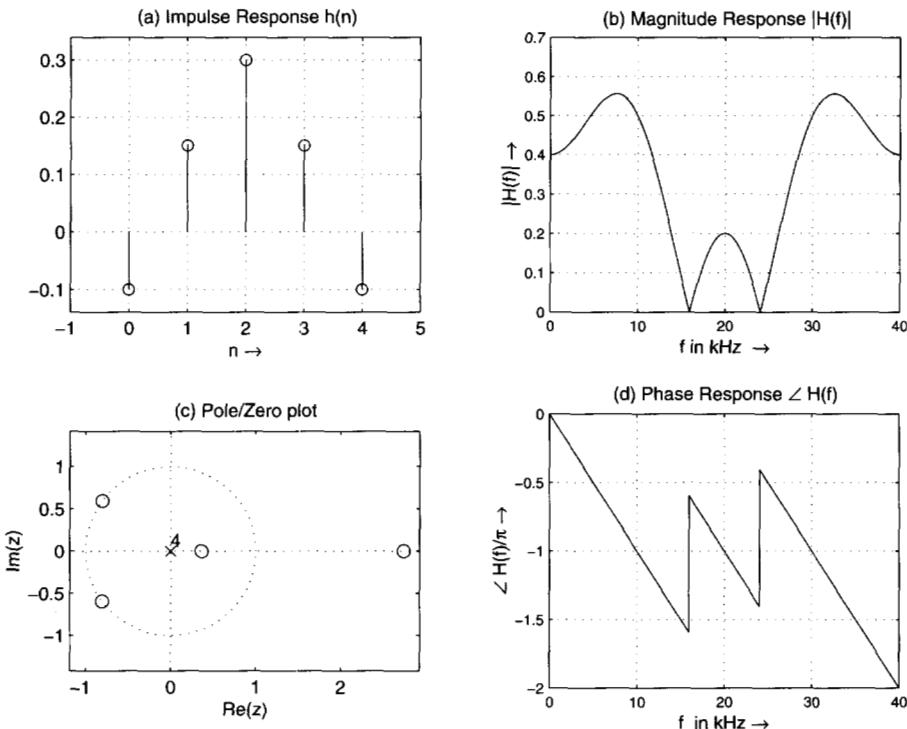


Figure 1.24 FIR system: (a) impulse response, (b) magnitude response, (c) pole/zero plot and (d) phase response (sampling frequency $f_s = 40$ kHz).

```
M-file 1.14 (figure1_24.m)
function magphasresponse(h)
FS=40000;
FoSi='Fontsize';
fosi=10;
if nargin==0
    h=[-.1 .15 .3 .15 -.1];
end
hmax=max(h);
hmin=min(h);
dh=hmax-hmin;
hmax=hmax+.1*dh;
hmin=hmin-.1*dh;

N=length(h);
% denominator polynomial:
a=zeros(1,N);
a(1)=1;

subplot(221)
stem(0:N-1,h)
axis([-1 N, hmin hmax])
title('a) Impulse Response h(n)',FoSi,fosi);
xlabel('n \rightarrow',FoSi,fosi)
grid on;

subplot(223)
zplane(h,a)
title('c) Pole/Zero plot',FoSi,fosi);
xlabel('Re(z)',FoSi,fosi)
ylabel('Im(z)',FoSi,fosi)

subplot(222)
[H,F] =freqz(h,a,1024,'whole',FS);
plot(F/1000,abs(H))
xlabel('f in kHz \rightarrow',FoSi,fosi);
ylabel('|H(f)| \rightarrow',FoSi,fosi);
title('b) Magnitude response |H(f)|',FoSi,fosi);
grid on;

subplot(224)
plot(F/1000,unwrap(angle(H))/pi)
xlabel('f in kHz \rightarrow',FoSi,fosi)
ylabel('\angle H(f)/\pi \rightarrow',FoSi,fosi)
title('d) Phase Response \angle H(f)',FoSi,fosi);
grid on;
```

1.3 Conclusion

In this first chapter some basic concepts of digital signals, their spectra and digital systems have been introduced. The description is intended for persons with little or no knowledge of digital signal processing. The inclusion of MATLAB M-files for all stages of processing may serve as a basis for further programming in the following chapters. As well as showing simple tools for graphical representations of digital audio signals we have calculated the spectrum of a signal $x(n)$ by the use of the FFT M-file

- `Xmagnitude=abs(fft(x))`
`Xphase=angle(fft(x)).`

Time-domain processing for DAFX can be performed by block-based input-output computations which are based on the convolution formula (if the impulse response of a system is known) or difference equations (if the coefficients a and b are known). The computations can be done by the following M-files:

- `y=conv(h,x) %length of output signal l_y =l_h +l_x -1`
`y=filter(b,a,x) %l_y =l_x`

These M-files deliver an output vector containing the output signal $y(n)$ in a vector of corresponding length. Of course, these block processing algorithms perform their inner computations on a sample-by-sample basis. Therefore, we have also shown an example for the sample-by-sample programming technique, which can be modified according to different applications:

- `y=dafxalgorithm(parameters,x)`
`% Sample-by sample algorithm y(n)=function(parameters,x(n))`
`for n=1:length(x),`
`y(n)=....do something algorithm with x(n) and parameters;`
`end;`

That is all we need for DAFX exploration and programming, good luck!

Bibliography

- [Arf98] D. Arfib. Different ways to write digital audio effects programs. In *Proc. DAFX-98 Digital Audio Effects Workshop*, pp. 188–191, Barcelona, November 1998.
- [Arf99] D. Arfib. Visual representations for digital audio effects and their control. In *Proc. DAFX-99 Digital Audio Effects Workshop*, pp. 63–68, Trondheim, December 1999.

- [ME93] C. Marvin and G. Ewers. *A Simple Approach to Digital Signal Processing*. Texas Instruments, 1993.
- [Mit01] S.K Mitra. *Digital Signal Processing – A Computer-Based Approach*. McGraw-Hill, 2nd edition, 2001.
- [MSY98] J. McClellan, R. Schafer, and M. Yoher. *DSP FIRST: A Multimedia Approach*. Prentice-Hall, 1998.
- [Orf96] S.J. Orfanidis. *Introduction to Signal Processing*. Prentice-Hall, 1996.
- [Zöl97] U. Zölzer. *Digital Audio Signal Processing*. John Wiley & Sons, Ltd, 1997.

Chapter 2

Filters

P. Dutilleux, U. Zölzer

2.1 Introduction

The term filter can have a large number of different meanings. In general it can be seen as a way to select certain elements with desired properties from a larger set. Let us focus on the particular field of digital audio effects and consider a signal in the frequency domain. The signal can be seen as a set of partials having different frequencies and amplitudes. The filter will perform a selection of the partials according to the frequencies that we want to reject, retain or emphasize. In other words: the filter will modify the amplitude of the partials according to their frequency. Once implemented, it will turn out that this filter is a linear transformation. As an extension, linear transformations can be said to be filters. According to this new definition of a filter, any linear operation could be said to be a filter but this would go far beyond the scope of digital audio effects. It is possible to demonstrate what a filter is by using one's voice and vocal tract. Utter a vowel, a for example, at a fixed pitch and then utter other vowels at the same pitch. By doing that we do not modify our vocal cords but we modify the volume and the interconnection pattern of our vocal tract. The vocal cords produce a signal with a fixed harmonic spectrum whereas the cavities act as acoustic filters to enhance some portions of the spectrum. We have described filters in the frequency domain here because it is the usual way to consider them but they also have an effect in the time domain. After introducing a filter classification in the frequency domain, we will review typical implementation methods and the associated effects in the time domain.

The various types of filters can be defined according to the following classification:

- **Lowpass (LP)** filters select low frequencies up to the cut-off frequency f_c and attenuate frequencies higher than f_c .

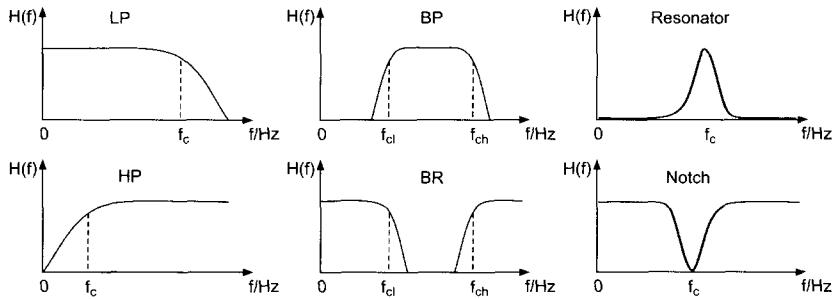


Figure 2.1 Filter classification.

- **Highpass (HP)** filters select frequencies higher than f_c and attenuate frequencies below f_c .
- **Bandpass (BP)** filters select frequencies between a lower cut-off frequency f_{cl} and a higher cut-off frequency f_{ch} . Frequencies below f_{cl} and frequencies higher than f_{ch} are attenuated.
- **Bandreject (BR)** filters attenuate frequencies between a lower cut-off frequency f_{cl} and a higher cut-off frequency f_{ch} . Frequencies below f_{cl} and frequencies higher than f_{ch} are passed.
- **Notch** filters attenuate frequencies in a narrow bandwidth around the cut-off frequency f_c .
- **Resonator** filters amplify frequencies in a narrow bandwidth around the cut-off frequency f_c .
- **Allpass** filters pass all frequencies but modify the phase of the input signal.

Other types of filters (LP with resonance, comb, multiple notch...) can be described as a combination of these basic elements. Here are listed some of the possible applications of these filter types: The lowpass with resonance is very often used in computer music to simulate an acoustical resonating structure; the highpass filter can remove undesired very low frequencies; the bandpass can produce effects such as the imitation of a telephone line or of a mute on an acoustical instrument; the bandreject can divide the audible spectrum into two bands that seem to be uncorrelated. The resonator can be used to add artificial resonances to a sound; the notch is most useful in eliminating annoying frequency components; a set of notch filters, used in combination with the input signal, can produce a phasing effect.

2.2 Basic Filters

2.2.1 Lowpass Filter Topologies

A filter can be implemented in various ways. It can be an acoustic filter, as in the case of the voice. For our applications we would rather use electronic or digital means. Although we are interested in digital audio effects, it is worth having a look at well-established analog techniques because a large body of methods have been developed in the past to design and build analog filters. There are intrinsic design methods for digital filters but many structures can be adapted from existing analog designs. Furthermore, some of them have been tailored for ease of operation within musical applications. It is therefore of interest to gain ideas from these analog designs in order to build digital filters having similar advantages. We will focus on the second-order lowpass filter because it is the most common type and other types can be derived from it. The frequency response of a lowpass filter is shown in Fig. 2.2. The tuning parameters of this lowpass filter are the cut-off frequency f_c and the damping factor ζ . The lower the damping factor, the higher the resonance at the cut-off frequency.

Analog Design, Sallen & Key

Let us remind ourselves of an analog circuit that implements a second-order lowpass filter with the least number of components: the Sallen & Key filter (Figure 2.2).

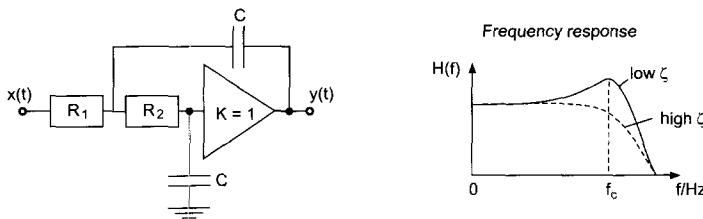


Figure 2.2 Sallen & Key second-order analog filter and frequency response.

The components (R_1, R_2, C) are related to the tuning parameters as:

$$f_c = \frac{1}{2\pi C \sqrt{R_1 R_2}} \quad \zeta = \frac{R_1 + R_2}{2\sqrt{R_1 R_2}} \quad (2.1)$$

These relations are straightforward but both tuning coefficients are coupled. It is therefore difficult to vary one while the other remains constant. This structure is therefore not recommended when the parameters are to be tuned dynamically and when low damping factors are desired.

Digital Design, Canonical

The canonical second-order structure, as shown in Fig. 2.3, can be implemented by the difference equation

$$\begin{aligned} y(n) = & b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) \\ & -a_1 y(n-1) - a_2 y(n-2). \end{aligned} \quad (2.2)$$

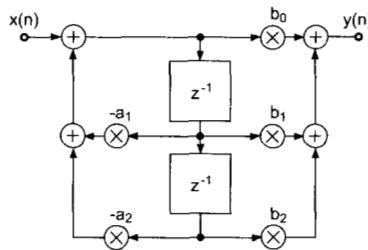


Figure 2.3 Canonical second-order digital filter.

It can be used for any second-order transfer function according to

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}. \quad (2.3)$$

In order to modify the cut-off frequency or the damping factor, all 5 coefficients have to be modified. They can be computed from the specification in the frequency plane or from a prototype analog filter. One of the methods that can be used is based on the bilinear transform [DJ85]. The following set of formulas compute the coefficients for a lowpass filter:

$$\begin{aligned} f_c & \text{analog cut-off frequency} \\ \zeta & \text{damping factor} \\ f_s & \text{sampling frequency} \\ C & = 1/[\tan(\pi f_c/f_s)] \end{aligned} \quad (2.4)$$

$$\begin{aligned} b_0 & = 1/(1 + 2\zeta C + C^2) \\ b_1 & = 2b_0 \\ b_2 & = b_0 \\ a_1 & = 2b_0(1 - C^2) \\ a_2 & = b_0(1 - 2\zeta C + C^2) \end{aligned} \quad (2.5)$$

This structure has the advantage that it requires very few elementary operations to process the signal itself. It has unfortunately some severe drawbacks. Modifying

the filter tuning (f_c, ζ) involves rather complex computations. If the parameters are varied continuously, the complexity of the filter is more dependent on the coefficient computation than on the filtering process itself. Another drawback is the poor signal to noise ratio for low frequency signals. Other filter structures are available that cope with these problems. We will again review a solution in the analog domain and its counterpart in the digital domain.

State Variable Filter, Analog

For musical applications of filters one wishes to have an independent control over the cut-off frequency and the damping factor. A technique originating from the analog computing technology can solve our problem. It is called the state variable filter (Figure 2.4). This structure is more expensive than the Sallen & Key but has independent tuning components (R_f, R_ζ) for the cut-off frequency and the damping factors:

$$f_c = 1/(2\pi R_f C) \quad \zeta = R/[2(R + R_\zeta)] \quad (2.6)$$

Furthermore, it provides simultaneously three types of outputs: lowpass, highpass and bandpass.

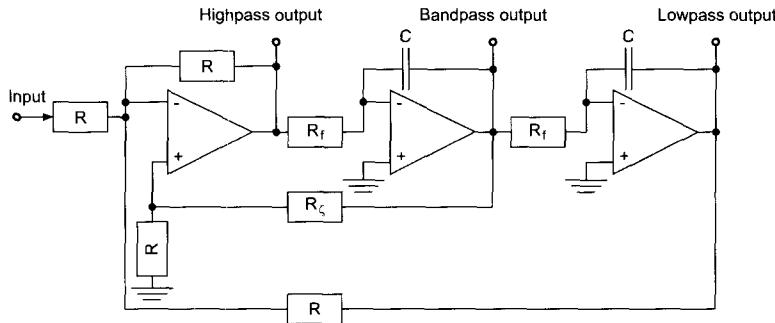


Figure 2.4 Analog state variable filter.

State Variable Filter, Digital

The state variable filter has a digital implementation, as shown in Fig. 2.5 [Cha80], where

$x(n)$	input signal
$y_l(n)$	lowpass output
$y_b(n)$	bandpass output
$y_h(n)$	highpass output

and the difference equations for the output signals are given by

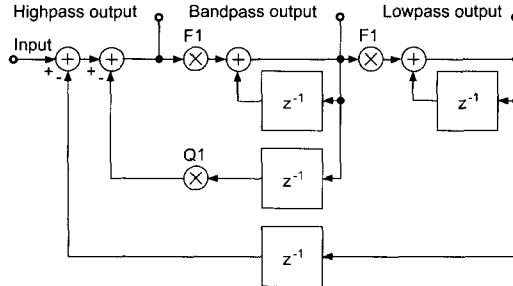


Figure 2.5 Digital state variable filter.

$$\begin{aligned} y_l(n) &= F_1 y_b(n) + y_l(n-1) \\ y_b(n) &= F_1 y_h(n) + y_b(n-1) \\ y_h(n) &= x(n) - y_l(n-1) - Q_1 y_b(n-1). \end{aligned} \quad (2.7)$$

With tuning coefficients F_1 and Q_1 , related to the tuning parameters f_c and ζ as:

$$F_1 = 2 \sin(\pi f_c / f_s) \quad Q_1 = 2\zeta \quad (2.8)$$

it can be shown that the lowpass transfer function is:

$$\begin{aligned} r &= F_1 \quad q = 1 - F_1 Q_1 \\ H(z) &= \frac{r^2}{1 + (r^2 - q - 1)z^{-1} + qz^{-2}} \end{aligned} \quad (2.9)$$

This structure is particularly effective not only as far as the filtering process is concerned but above all because of the simple relations between control parameters and tuning coefficients. One should consider the stability of this filter, because at higher cut-off frequencies and larger damping factors it becomes unstable. A “usability limit” given by $F_1 < 2 - Q_1$ assures the stable operation of the state variable implementation [Dut91, Die00]. In most musical applications however it is not a problem because the tuning frequencies are usually small compared to the sampling frequency and the damping factor is usually set to small values [Dut89a, Dut97]. This filter has proven its suitability for a large number of applications. The nice properties of this filter have been exploited to produce endless glissandi out of natural sounds and to allow smooth transitions between extreme settings [Dut89b, m-Vas93]. It is also used for synthesizer applications [Die00]. We have considered here two different digital filter structures. More are available and each has its advantages and drawbacks. An optimum choice can only be made in agreement with the application [Zöl97].

Normalization

Filters are usually designed in the frequency domain and we have seen that they have an action also in the time domain. Another correlated impact lies in the loudness of the filtered sounds. The filter might produce the right effect but the result

might be useless because the sound has become too weak or too strong. The method of compensating for these amplitude variations is called normalization. Usual normalization methods are called L_1 , L_2 and L_∞ [Zöl97]. L_1 is used when the filter should never be overloaded under any circumstances. This is overkill most of the time. L_2 is used to normalize the loudness of the signal. It is accurate for broad-band signals and fits many practical musical applications. L_∞ actually normalizes the frequency response. It is best when the signal to filter is sinusoidal or periodical. With a suitable normalization scheme the filter can prove to be very easy to handle whereas with the wrong normalization, the filter might be rejected by musicians because they cannot operate it. The normalization of the state variable filter has been studied in [Dut91] where several implementation schemes are proposed that lead to an effective implementation. In practice, a first-order lowpass filter that processes the input signal will perform the normalization in f_c and an amplitude correction in $\sqrt{\zeta}$ will normalize in ζ (Figure 2.6). This normalization scheme allows us to operate the filter with damping factors down to 10^{-4} where the filter gain reaches about 74 dB at f_c .

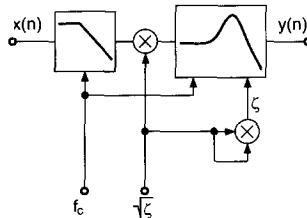


Figure 2.6 L_2 -normalization in f_c and ζ for the state variable filter.

Sharp Filters

Apart from FIR filters (see section 2.2.3), we have so far only given examples of second-order filters. These filters are not suitable for all applications. On the one hand, smooth spectral modifications are better realized by using first-order filters. On the other hand, processing two signal components differently that are close in frequency, or imitating the selectivity of our hearing system calls for higher order filters. FIR filters can offer the right selectivity but again, they will not be easily tuned. Butterworth filters have attractive features in this case. Such filters are optimized for a flat frequency response until f_c and yield a $6n$ dB/octave attenuation for frequencies higher than f_c . Filters of order $2n$ can be built out of n second-order sections. All sections are tuned to the same cut-off frequency f_c but each section has a different damping factor ζ (Table 2.1) [LKG72].

These filters can be implemented accurately in the canonical second-order digital filter structure but modifying the tuning frequency in real time can lead to temporary instabilities. The state variable structure is less accurate for high tuning frequencies (i.e. $f_c > f_s/10$) but allows faster tuning modifications. A bandpass filter comprising a 4th-order highpass and a 4th-order lowpass was implemented

Table 2.1 Damping factors for Butterworth filters.

n	ζ of second-order sections				
2	0.707				
4	0.924	0.383			
6	0.966	0.707	0.259		
8	0.981	0.831	0.556	0.195	
10	0.988	0.891	0.707	0.454	0.156

and used to imitate a fast varying mute on a trombone [Dut91]. Higher order filters (up to about 16) are useful to segregate spectral bands or even individual partials within complex sounds.

Behavior in the Time Domain

We so far considered the action of the filters in the frequency domain. We cannot forget the time domain because it is closely related to it. Narrow bandpass filters, or resonant filters even more, will induce long ringing time responses. Filters can be optimized for their frequency response or time response. It is easier to grasp the time behavior of FIRs than IIRs. FIRs have the drawback of a time delay that can impair the responsiveness of digital audio effects.

2.2.2 Parametric AP, LP, HP, BP and BR Filters

Introduction

In this subsection we introduce a special class of parametric filter structures for allpass, lowpass, highpass, bandpass and bandreject filter functions. Parametric filter structures denote special signal flow graphs where a coefficient inside the signal flow graph directly controls the cut-off frequency and bandwidth of the corresponding filter. These filter structures are easily tunable by changing only one or two coefficients. They play an important role for real-time control with minimum computational complexity.

Signal Processing

The basis for parametric first- and second-order IIR filters is the first- and second-order allpass filter. We will first discuss the first-order allpass and show simple low- and highpass filters, which consist of a tunable allpass filter together with a direct path.

First-order allpass. A first-order allpass filter is given by the transfer function

$$A(z) = \frac{z^{-1} + c}{1 + cz^{-1}} \quad (2.10)$$

$$c = \frac{\tan(\pi f_c/f_s) - 1}{\tan(\pi f_c/f_s) + 1}. \quad (2.11)$$

The magnitude/phase response and the group delay of a first-order allpass are shown in Fig. 2.7. The magnitude response is equal to one and the phase response is approaching -180 degrees for high frequencies. The group delay shows the delay of the input signal in samples versus frequency. The coefficient c in (2.10) controls the cut-off frequency of the allpass, where the phase response passes -90 degrees (see Fig. 2.7).

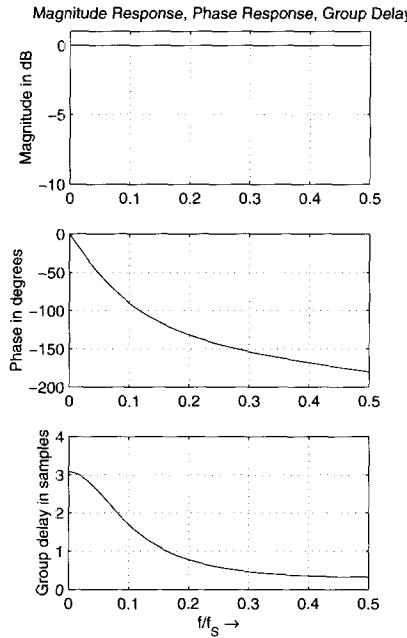


Figure 2.7 First-order allpass filter with $f_c = 0.1 \cdot f_s$.

From (2.10) we can derive the corresponding difference equation

$$y(n) = cx(n) + x(n-1) - cy(n-1), \quad (2.12)$$

which leads to the block diagram in Fig. 2.8. The coefficient c occurs twice in this signal flow graph and can be adjusted according to (2.11) to change the cut-off frequency. A variant allpass structure with only one delay element is shown in the right part of Fig. 2.8. It is implemented by the difference equations

$$x_h(n) = x(n) - cx_h(n-1) \quad (2.13)$$

$$y(n) = cx_h(n) + x_h(n-1). \quad (2.14)$$

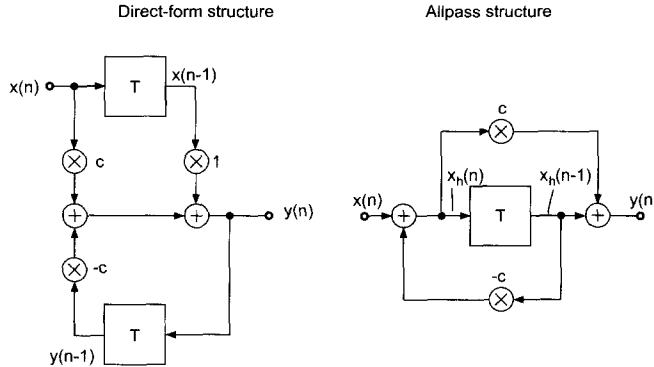


Figure 2.8 Block diagram for a first-order allpass filter.

The resulting transfer function is equal to (2.10). For simple implementations a table with a number of coefficients for different cut-off frequencies is sufficient, but even for real-time applications this structure offers very few computations. In the following we use this first-order allpass filter to perform low/highpass filtering.

First-order low/highpass. A first-order lowpass filter can be achieved by adding or subtracting (+/-) the input signal from the output signal of a first-order allpass filter. As the output signal of the first-order allpass filter has a phase shift of -180 degrees for high frequencies, this operation leads to low/highpass filtering. The transfer function of a low/highpass filter is then given by

$$H(z) = \frac{1}{2} (1 \pm A(z)) \quad (\text{LP/HP} +/-) \quad (2.15)$$

$$A(z) = \frac{z^{-1} + c}{1 + cz^{-1}} \quad (2.16)$$

$$c = \frac{\tan(\pi f_c/f_s) - 1}{\tan(\pi f_c/f_s) + 1}, \quad (2.17)$$

where a tunable first-order allpass $A(z)$ with tuning parameter c is used. The plus sign (+) denotes the lowpass operation and the minus sign (-) the highpass operation. A block diagram in Fig. 2.9 represents the operations involved in performing the low/highpass filtering. The allpass filter can be implemented by the difference equation (2.12) as shown in Fig. 2.8.

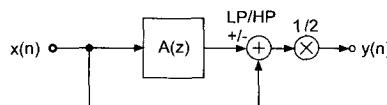


Figure 2.9 Block diagram of a first-order low/highpass filter.

The magnitude/phase response and group delay are illustrated for low- and high-pass filtering in Fig. 2.10. The -3dB point of the magnitude response for lowpass and

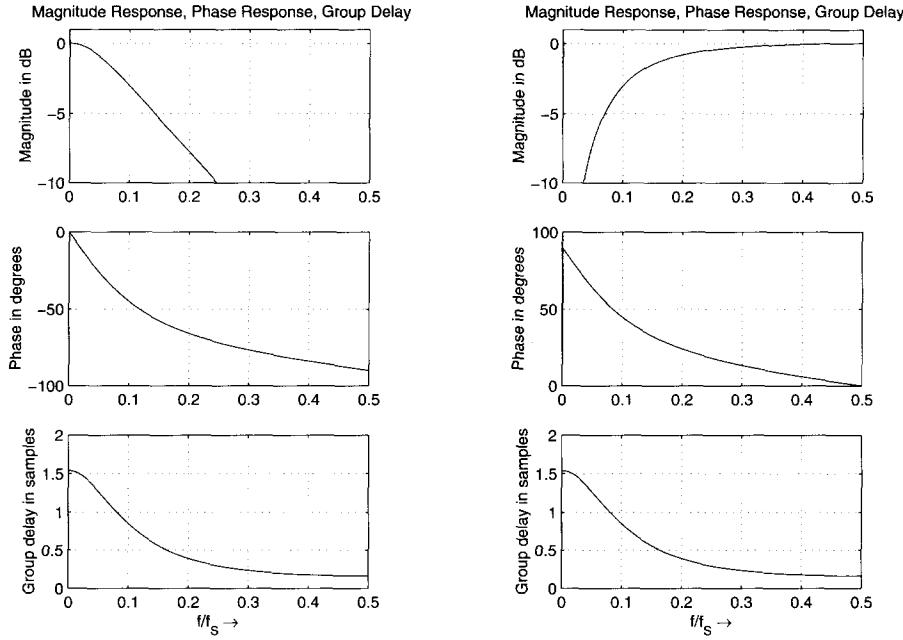


Figure 2.10 First-order low/highpass filter with $f_c = 0.1f_s$.

highpass is passed at the cut-off frequency. With the help of the allpass subsystem in Fig. 2.9 tunable low- and highpass systems are achieved.

Second-order allpass. The implementation of tunable bandpass and band-reject filters can be achieved with a second-order allpass filter. The transfer function of a second-order allpass filter is given by

$$A(z) = \frac{-c + d(1 - c)z^{-1} + z^{-2}}{1 + d(1 - c)z^{-1} - cz^{-2}} \quad (2.18)$$

$$c = \frac{\tan(\pi f_b/f_s) - 1}{\tan(\pi f_b/f_s) + 1} \quad (2.19)$$

$$d = -\cos(2\pi f_c/f_s). \quad (2.20)$$

The parameter d adjusts the cut-off frequency and the parameter c the bandwidth. The magnitude/phase response and the group delay of a second-order allpass are shown in Fig. 2.7. The magnitude response is again equal to one and the phase response approaches -360 degrees for high frequencies. The cut-off frequency ω_C determines the point on the phase curve, where the phase response passes -180 degrees. The width or slope of the phase transition around the cut-off frequency is controlled by the bandwidth parameter ω_B . From (2.18) the corresponding difference equation

$$\begin{aligned} y(n) = & -cx(n) + d(1 - c)x(n - 1) + x(n - 2) \\ & -d(1 - c)y(n - 1) + cy(n - 2) \end{aligned} \quad (2.21)$$

can be derived, which leads to the block diagram in Fig. 2.12. The cut-off frequency is controlled by the coefficient d and the bandwidth by coefficient c .

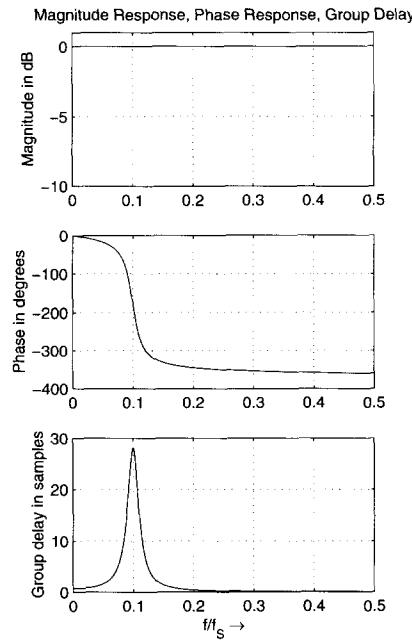


Figure 2.11 Second-order allpass filter with $f_c = 0.1f_s$ and $f_b = 0.022f_s$.

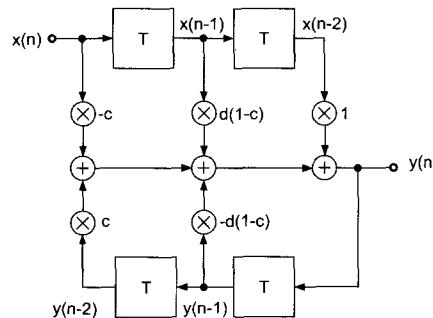


Figure 2.12 Block diagram for a second-order allpass filter.

Second-order bandpass/bandreject. Second-order bandpass and bandreject filters can be described by the following transfer function

$$H(z) = \frac{1}{2} [1 \mp A(z)] \quad (\text{BP/BR } -/+ \quad (2.22)$$

$$A(z) = \frac{-c + d(1 - c)z^{-1} + z^{-2}}{1 + d(1 - c)z^{-1} - cz^{-2}} \quad (2.23)$$

$$c = \frac{\tan(\pi f_b/f_s) - 1}{\tan(2\pi f_b/f_s) + 1} \quad (2.24)$$

$$d = -\cos(2\pi f_c/f_s), \quad (2.25)$$

where a tunable second-order allpass $A(z)$ with tuning parameters c and d is used. The plus sign (+) denotes the bandpass operation and the minus sign (-) the bandreject operation. The block diagram in Fig. 2.13 shows the bandpass and bandreject filter implementation based on a second-order allpass subsystem, which can be implemented by the signal flow graph of Fig. 2.12. The magnitude/phase response and group delay are illustrated in Fig. 2.14 for both filter types.

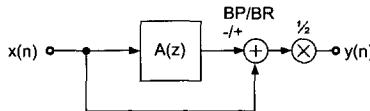


Figure 2.13 Second-order bandpass and bandreject filter.

Second-order low/highpass filters. The coefficients for second-order low- and highpass filters given by the transfer function of (2.3) are shown in Table 2.2. A control of single coefficients for adjusting the cut-off frequency is not possible. A complete set of coefficients is necessary, if the cut-off frequency is changed. The implementation of these second-order low- and highpass filters can be achieved by the difference equation (2.2) and the filter structure in Fig. 2.3.

Table 2.2 Filter coefficients for second-order lowpass/highpass filters [Zöl97].

lowpass (second-order) with $K = \tan(\pi f_c/f_s)$				
b_0	b_1	b_2	a_1	a_2
$\frac{K^2}{1+\sqrt{2}K+K^2}$	$\frac{2K^2}{1+\sqrt{2}K+K^2}$	$\frac{K^2}{1+\sqrt{2}K+K^2}$	$\frac{2(K^2-1)}{1+\sqrt{2}K+K^2}$	$\frac{1-\sqrt{2}K+K^2}{1+\sqrt{2}K+K^2}$
highpass (second-order) with $K = \tan(\pi f_c/f_s)$				
b_0	b_1	b_2	a_1	a_2
$\frac{1}{1+\sqrt{2}K+K^2}$	$\frac{-2}{1+\sqrt{2}K+K^2}$	$\frac{1}{1+\sqrt{2}K+K^2}$	$\frac{2(K^2-1)}{1+\sqrt{2}K+K^2}$	$\frac{1-\sqrt{2}K+K^2}{1+\sqrt{2}K+K^2}$

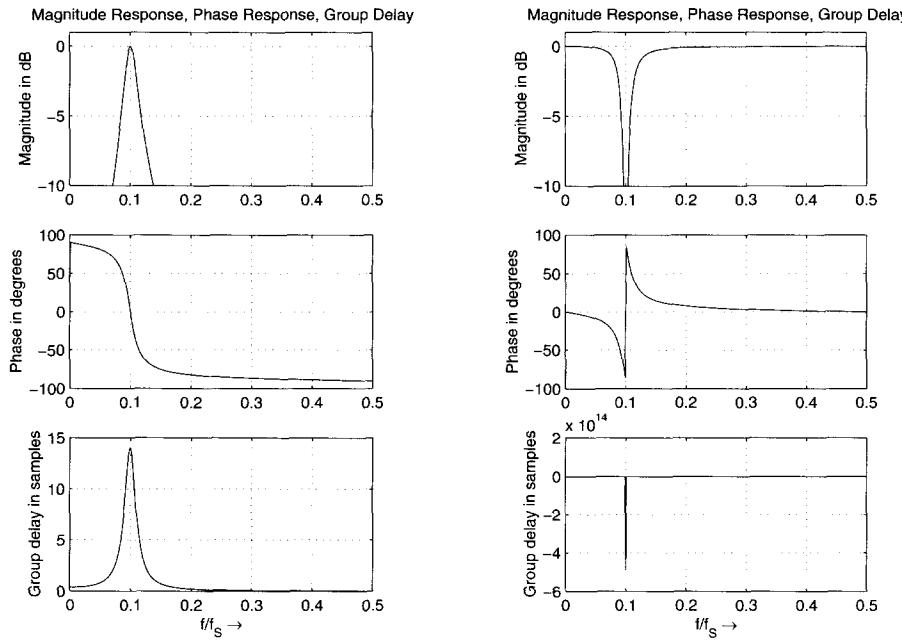


Figure 2.14 Second-order bandpass/bandreject filter with $f_c = 0.1f_s$ and $f_b = 0.022f_s$.

Series connection of first- and second-order filters. If several filters are necessary for spectrum shaping, a series connection of first- and second-order filters

$$H_{\text{1st-order}}(z) = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1}} \quad (2.26)$$

$$H_{\text{2nd-order}}(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \quad (2.27)$$

is performed, which is given by the product of the single transfer functions

$$H(z) = \frac{Y(z)}{X(z)} = H_1(z) \cdot H_2(z) \cdot H_3(z). \quad (2.28)$$

A series connection of three stages is shown in Fig. 2.15. The resulting difference

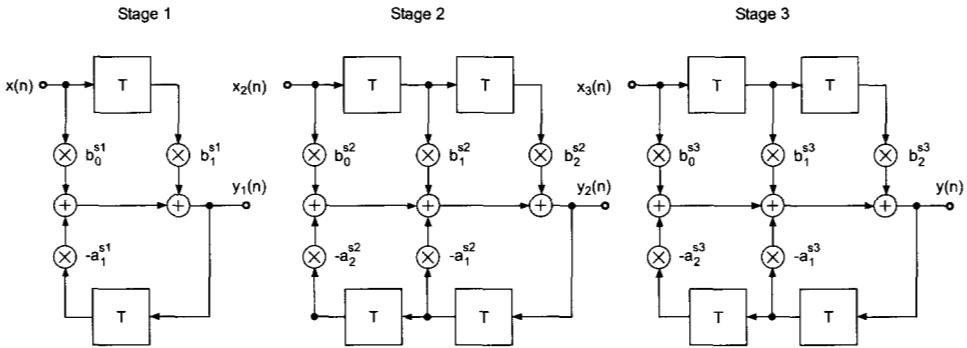


Figure 2.15 Series connection of first/second-order stages.

equation can be split into three difference equations as given by

stage 1

$$y_1(n) = b_0^{s1}x(n) + b_1^{s1}x(n-1) - a_1^{s1}y_1(n-1) \quad (2.29)$$

stage 2

$$x_2(n) = y_1(n) \quad (2.30)$$

$$\begin{aligned} y_2(n) &= b_0^{s2}x_2(n) + b_1^{s2}x_2(n-1) + b_2^{s2}x_2(n-2) \\ &\quad - a_1^{s2}y_2(n-1) - a_2^{s2}y_2(n-2) \end{aligned} \quad (2.31)$$

stage 3

$$x_3(n) = y_2(n) \quad (2.32)$$

$$\begin{aligned} y(n) &= b_0^{s3}x_3(n) + b_1^{s3}x_3(n-1) + b_2^{s3}x_3(n-2) \\ &\quad - a_1^{s3}y(n-1) - a_2^{s3}y(n-2). \end{aligned} \quad (2.33)$$

Musical Applications

The simple control of the cut-off frequency and the bandwidth of these parametric filters leads to very efficient implementations for real-time audio applications. Only second-order low- and highpass filters need the computation of a complete set of coefficients. The series connection of these filters can be done very easily as shown in the previous paragraph.

2.2.3 FIR Filters

Introduction

The digital filter that we have seen before is said to have an infinite impulse response. Because of the feedback loops within the structure, an input sample will excite an output signal whose duration is dependent on the tuning parameters and can extend over a fairly long period of time. There are other filter structures without

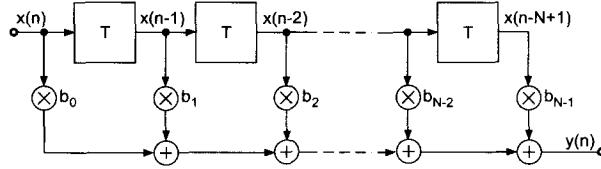


Figure 2.16 Finite Impulse Response Filter.

feedback loops (Figure 2.16). These are called finite impulse response filters (FIR), because the response of the filter to a unit impulse lasts only for a fixed period of time. These filters allow the building of sophisticated filter types where strong attenuation of unwanted frequencies or decomposition of the signal into several frequency bands is necessary. They typically require more computing power than IIR structures to achieve similar results but when they are implemented in the form known as fast convolution they become competitive, thanks to the FFT algorithm. It is rather unwieldy to tune these filters interactively. As an example, let us briefly consider the vocoder application. If the frequency bands are fixed, then the FIR implementation can be most effective but if the frequency bands have to be subtly tuned by a performer, then the IIR structures will certainly prove superior [Mai97]. However, the filter structure in Fig. 2.16 finds widespread applications for head-related transfer functions and the approximation of first room reflections, as will be shown in Chapter 6. For applications where the impulse response of a real system has been measured, the FIR filter structure can be used directly to simulate the measured impulse response.

Signal Processing

The output/input relation of the filter structure in Fig. 2.16 is described by the difference equation

$$y(n) = \sum_{i=0}^{N-1} b_i \cdot x(n-i) \quad (2.34)$$

$$= b_0 x(n) + b_1 x(n-1) + \cdots + b_{N-1} x(n-N+1), \quad (2.35)$$

which is a weighted sum of delayed input samples. If the input signal is a unit impulse $\delta(n)$, which is one for $n = 0$ and zero for $n \neq 0$, we get the impulse response of the system according to

$$h(n) = \sum_{i=0}^{N-1} b_i \cdot \delta(n-i). \quad (2.36)$$

A graphical illustration of the impulse response of a 5-tap FIR filter is shown in Fig. 2.17. The Z-transform of the impulse response gives the transfer function

$$H(z) = \sum_{i=0}^{N-1} b_i \cdot z^{-i} \quad (2.37)$$

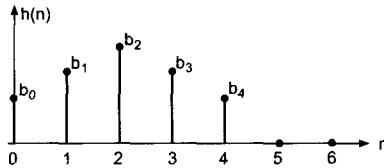


Figure 2.17 Impulse response of an FIR filter.

and with $z = e^{j\Omega}$ the frequency response

$$\begin{aligned} H(e^{j\Omega}) &= b_0 + b_1 e^{-j\Omega} + b_2 e^{-j2\Omega} + \cdots + b_{N-1} \cdot e^{-j(N-1)\Omega} \\ \text{with } \Omega &= 2\pi f/f_s = \omega T. \end{aligned} \quad (2.38)$$

Filter design. The filters already described such as LP, HP, BP and BR are also possible with FIR filter structures (see Fig. 2.18). The N coefficients b_0, \dots, b_{N-1} of a nonrecursive filter have to be computed by special design programs, which are discussed in all DSP text books. The N coefficients of the impulse response can be designed to yield a linear phase response, when the coefficients fulfill certain symmetry conditions. The simplest design is based on the inverse discrete-time Fourier transform of the ideal lowpass filter, which leads to the impulse response

$$h(n) = \frac{2f_c}{f_s} \cdot \frac{\sin [2\pi f_c/f_s (n - \frac{N-1}{2})]}{2\pi f_c/f_s (n - \frac{N-1}{2})}, n = 0, \dots, N-1. \quad (2.39)$$

To improve the frequency response this impulse response can be weighted by an appropriate window function like Hamming or Blackman according to

$$h_B(n) = h(n) \cdot w_B(n) \quad (2.40)$$

$$h_H(n) = h(n) \cdot w_H(n) \quad (2.41)$$

$$n = 0, 1, \dots, N-1.$$

If a lowpass filter is designed and an impulse response $h_{LP}(n)$ is derived, a *frequency transformation* of this lowpass filter leads to highpass, bandpass and bandreject filters (see Fig. 2.18).

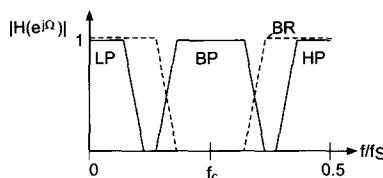


Figure 2.18 Frequency transformations: LP and frequency transformations to BP and HP.

Frequency transformations are performed in the time domain by taking the lowpass impulse response $h_{LP}(n)$ and computing the following equations:

- LP-HP

$$h_{HP}(n) = h_{LP}(n) \cdot \cos \left[\pi \left(n - \frac{N-1}{2} \right) \right] \quad n = 0, \dots, N-1 \quad (2.42)$$

- LP-BP

$$h_{BP}(n) = 2h_{LP}(n) \cdot \cos \left[2\pi \frac{f_c}{f_s} \left(n - \frac{N-1}{2} \right) \right] \quad n = 0, \dots, N-1 \quad (2.43)$$

- LP-BR

$$h_{BR}(n) = \delta \left(n - \frac{N-1}{2} \right) - h_{BP}(n) \quad n = 0, \dots, N-1. \quad (2.44)$$

Another simple FIR filter design is based on the FFT algorithm and is called *frequency sampling*. Design examples for audio processing with this design technique can be found in [Zöl97].

Musical Applications

If linear phase processing is required, FIR filtering offers magnitude equalization without phase distortions. They allow real-time equalization by making use of the frequency sampling design procedure [Zöl97] and are attractive equalizer counterparts to IIR filters, as shown in [McG93]. A discussion of more advanced FIR filters for audio processing can be found in [Zöl97].

2.2.4 Convolution

Introduction

Convolution is a generic signal processing operation like addition or multiplication. In the realm of computer music it has nevertheless the particular meaning of imposing a spectral or temporal structure onto a sound. These structures are usually not defined by a set of few parameters, such as the shape or the time response of a filter, but given by a signal which lasts typically a few seconds or more. Although convolution has been known and used for a very long time in the signal processing community, its significance for computer music and audio processing has grown with the availability of fast computers that allow long convolutions to be performed in a reasonable period of time.

Signal Processing

We could say in general that the convolution of two signals means filtering the one with the other. There are several ways of performing this operation. The straightforward method is a direct implementation in a FIR filter structure but it is computationally very ineffective when the impulse response is several thousand samples long. Another method, called the fast convolution, makes use of the FFT algorithm to dramatically speed up the computation. The drawback of the fast convolution is that it has a processing delay equal to the length of two FFT blocks, which is objectionable for real-time applications whereas the FIR method has the advantage of providing a result immediately after the first sample has been computed. In order to take advantage of the FFT algorithm while keeping the processing delay to a minimum, low-latency convolution schemes have been developed which are suitable for real-time applications [Gar95, MT99].

The result of convolution can be interpreted in both the frequency and time domains. If $a(n)$ and $b(n)$ are the two convolved signals, the output spectrum will be given by the product of the two spectra $S(f) = A(f) \cdot B(f)$. The time interpretation derives from the fact that if $b(n)$ is a pulse at time k , we will obtain a copy of $a(n)$ shifted at time k_0 , i.e. $s(n) = a(n - k)$. If $b(n)$ is a sequence of pulses, we will obtain a copy of $a(n)$ in correspondence to every pulse, i.e. a rhythmic, pitched, or reverberated structure, depending on the pulse distance. If $b(n)$ is pulse-like, we obtain the same pattern with a filtering effect. In this case $b(n)$ should be interpreted as an impulse response. Thus convolution will result in subtractive synthesis, where the frequency shape of the filter is determined by a real sound. For example the convolution with a bell sound will be heard as filtered by the resonances of the bell. In fact the bell sound is generated by a strike on the bell and can be considered as the impulse response of the bell. In this way we can simulate the effect of a sound hitting a bell, without measuring the resonances and designing the filter. If both sounds $a(n)$ and $b(n)$ are complex in time and frequency, the resulting sound will be blurred and will tend to lack the original sound's character. If both sounds are of long duration and each has a strong pitch and smooth attack, the result will contain both pitches and the intersection of their spectra.

Musical Applications

The sound example “quasthal” [m-quasthal] illustrates the use of the impulse response as a way of characterizing a linear system. In this example, a spoken word is convolved with a series of impulses which are derived from measurements of 2 loudspeakers and of 3 rooms. The first loudspeaker, a small studio monitor, alters at least the original sound. The second loudspeaker, a spherical one, colors the sound strongly. When the sound is convolved with the impulse responses of a room, it is projected in the corresponding virtual auditory space [DMT99]. A diffuse reverberation can be produced by convolving with broad band noise having a sharp attack and exponentially decreasing amplitude. Another example features a tuba glissando convolved by a series of snare-drum strokes. The tuba is transformed in something like a tibetan trumpet playing in the mountains. Each stroke

of the snare drum produces a copy of the tuba sound. Since each stroke is noisy and broadband, it acts like a reverberator. The series of strokes acts like several diffusing boundaries and produces the type of echo that can be found in natural landscapes [DMT99, m-tubg5sna].

The convolution can be used to map a rhythm pattern onto a sampled sound. The rhythm pattern can be defined by positioning a unit impulse at each desired time within a signal block. The convolution of the input sound with the time pattern will produce copies of the input signal at each of the unit impulses. If the unit impulse is replaced by a more complex sound, each copy will be modified in its timbre and in its time structure. If a snare drum stroke is used, the attacks will be smeared and some diffusion will be added [m-gendsna]. The convolution has an effect both in the frequency and in the time domain. Take a speech sound with sharp frequency resonances and a rhythm pattern defined by a series of snare-drum strokes. Each word will appear with the rhythm pattern, also the rhythm pattern will be heard in each word with the frequency resonances of the initial speech sound [m-chu5sna].

The convolution as a tool for musical composition has been investigated by composers such as Horacio Vaggione [m-Vag96, Vag98] and Curtis Roads [Roa97]. Because the convolution has a combined effect in the time and frequency domains, some expertise is necessary to foresee the result of the combination of two sounds.

2.3 Equalizers

Introduction and Musical Applications

In contrast to low/highpass and bandpass/reject filters, which attenuate the audio spectrum above or below a cut-off frequency, equalizers shape the audio spectrum by enhancing certain frequency bands while others remain unaffected. They are built by a series connection of first- and second-order shelving and peak filters, which are controlled independently (see Fig. 2.19). Shelving filters boost or cut the low or high frequency bands with the parameter cut-off frequency f_c and gain G . Peak filters boost or cut mid-frequency bands with parameters cut-off frequency f_c , bandwidth f_b and gain G . One often used filter type is the constant Q peak filter. The Q factor is defined by the ratio of the bandwidth to cut-off frequency $Q = \frac{f_b}{f_c}$. The cut-off frequency of peak filters are then tuned, while keeping the Q factor constant. This means that the bandwidth is increased when the cut-off frequency is increased and vice versa. Several proposed digital filter structures for shelving and peak filters can be found in the literature [Whi86, RM87, Dut89a, HB93, Bri94, Orf96, Orf97, Zöl97].

Applications of these parametric filters can be found in parametric equalizers, octave equalizers ($f_c=31.25, 62.5, 125, 250, 500, 1000, 2000, 4000, 8000, 16000$ Hz) and all kinds of equalization devices in mixing consoles, outboard equipment and foot pedal controlled stomp boxes.

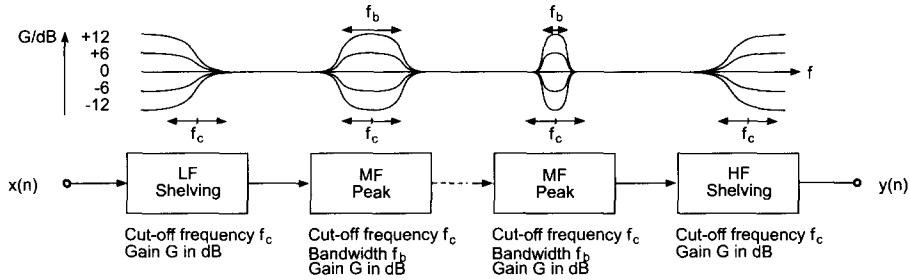


Figure 2.19 Series connection of shelving and peak filters.

2.3.1 Shelving Filters

First-order Design

First-order low/high frequency shelving filters [Zöl97] can be described by the transfer function

$$H(z) = 1 + \frac{H_0}{2} [1 + \pm A(z)] \quad (\text{LF/HF } +/-) \quad (2.45)$$

with the first-order allpass

$$A(z) = \frac{z^{-1} + a_{B/C}}{1 + a_{B/C}z^{-1}}. \quad (2.46)$$

The block diagram in Fig. 2.20 shows a first-order low/high-frequency shelving

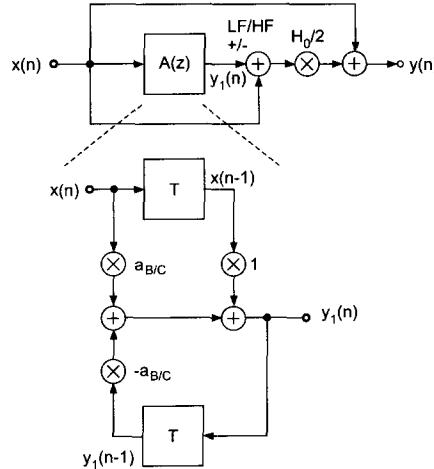


Figure 2.20 First-order low/high-frequency shelving filter.

filter, which leads to the following difference equations:

$$y_1(n) = a_{B/C}x(n) + x(n-1) - a_{B/C}y_1(n-1) \quad (2.47)$$

$$y(n) = \frac{H_0}{2} [x(n) \pm y_1(n)] + x(n). \quad (2.48)$$

The gain G in dB for low/high frequencies can be adjusted by the parameter

$$H_0 = V_0 - 1, \quad \text{with} \quad V_0 = 10^{G/20}. \quad (2.49)$$

The cut-off frequency parameter a_B for boost and a_C for cut can be calculated as

$$a_B = \frac{\tan((\pi f_c/f_s) - 1)}{\tan((\pi f_c/f_s) + 1)}, \quad \omega_c = 2\pi f_c \quad (2.50)$$

$$a_C = \frac{\tan((\pi f_c/f_s) - V_0)}{\tan((\pi f_c/f_s) + V_0)}. \quad (2.51)$$

The cut-off frequency parameters for boost and cut for a first-order high-frequency shelving filter [Zöl97] are calculated by

$$a_B = \frac{\tan(\pi f_c/f_s) - 1}{\tan((\pi f_c/f_s) + 1)} \quad (2.52)$$

$$a_C = \frac{V_0 \tan((\pi f_c/f_s) - 1)}{V_0 \tan((\pi f_c/f_s) + 1)}. \quad (2.53)$$

Magnitude responses for a low-frequency shelving filter are illustrated in the left part of Fig. 2.21 for several cut-off frequencies and gain factors. The slope of the frequency curves for these first-order filters are with 6 dB per octave.

Second-order Design

For several applications especially in advanced equalizer designs the slope of the shelving filter is further increased by second-order transfer functions. Design formulas for second-order shelving filters are given in Table 2.3 from [Zöl97]. Magnitude responses for second-order low/high frequency shelving filters are illustrated in the right part of Fig. 2.21 for two cut-off frequencies and several gain factors.

2.3.2 Peak Filters

A second-order peak filter [Zöl97] is given by the transfer function

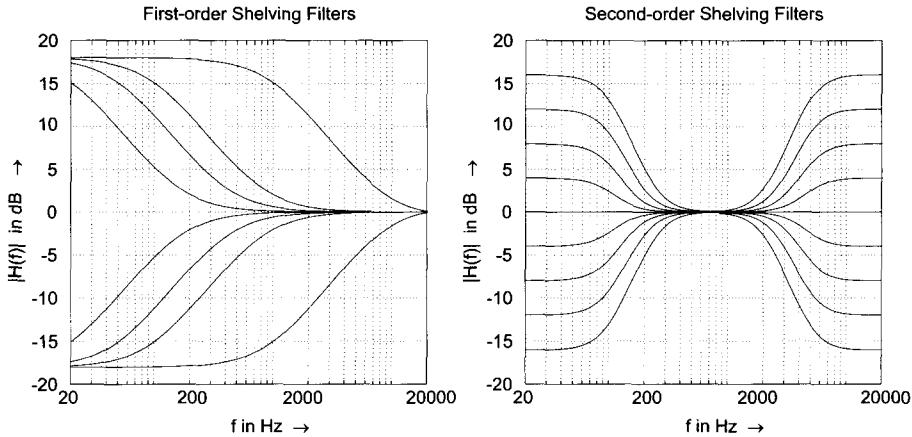
$$H(z) = 1 + \frac{H_0}{2} [1 - A_2(z)], \quad (2.54)$$

where

$$A_2(z) = \frac{-a_B + (d - da_B)z^{-1} + z^{-2}}{1 + (d - da_B)z^{-1} - a_B z^{-2}} \quad (2.55)$$

Table 2.3 Second-order shelving filter design with $K = \tan(\pi f_c/f_s)$ [Zöl97].

low-frequency shelving (boost $V_0 = 10^{G/20}$)				
b_0	b_1	b_2	a_1	a_2
$\frac{1+\sqrt{2}V_0K+V_0K^2}{1+\sqrt{2}K+K^2}$	$\frac{2(V_0K^2-1)}{1+\sqrt{2}K+K^2}$	$\frac{1-\sqrt{2}V_0K+V_0K^2}{1+\sqrt{2}K+K^2}$	$\frac{2(K^2-1)}{1+\sqrt{2}K+K^2}$	$\frac{1-\sqrt{2}K+K^2}{1+\sqrt{2}K+K^2}$
low-frequency shelving (cut $V_0 = 10^{-G/20}$)				
b_0	b_1	b_2	a_1	a_2
$\frac{1+\sqrt{2}V_0K+K^2}{1+\sqrt{2}V_0K+V_0K^2}$	$\frac{2(K^2-1)}{1+\sqrt{2}V_0K+V_0K^2}$	$\frac{1-\sqrt{2}K+K^2}{1+\sqrt{2}V_0K+V_0K^2}$	$\frac{2(V_0K^2-1)}{1+\sqrt{2}V_0K+V_0K^2}$	$\frac{1-\sqrt{2}V_0K+V_0K^2}{1+\sqrt{2}V_0K+V_0K^2}$
high-frequency shelving (boost $V_0 = 10^{G/20}$)				
b_0	b_1	b_2	a_1	a_2
$\frac{V_0+\sqrt{2}V_0K+K^2}{1+\sqrt{2}K+K^2}$	$\frac{2(K^2-V_0)}{1+\sqrt{2}K+K^2}$	$\frac{V_0-\sqrt{2}V_0K+K^2}{1+\sqrt{2}K+K^2}$	$\frac{2(K^2-1)}{1+\sqrt{2}K+K^2}$	$\frac{1-\sqrt{2}K+K^2}{1+\sqrt{2}K+K^2}$
high-frequency shelving (cut $V_0 = 10^{-G/20}$)				
b_0	b_1	b_2	a_1	a_2
$\frac{1+\sqrt{2}K+K^2}{V_0+\sqrt{2}V_0K+K^2}$	$\frac{2(K^2-1)}{V_0+\sqrt{2}V_0K+K^2}$	$\frac{1-\sqrt{2}K+K^2}{V_0+\sqrt{2}V_0K+K^2}$	$\frac{2(K^2/V_0-1)}{1+\sqrt{2}/V_0K+K^2/V_0}$	$\frac{1-\sqrt{2}/V_0K+K^2/V_0}{1+\sqrt{2}/V_0K+K^2/V_0}$

**Figure 2.21** Frequency responses for first-order and second-order shelving filters.

is a second-order allpass filter. The block diagram in Fig. 2.22 shows the second-order peak filter, which leads to the following difference equations:

$$\begin{aligned} y_1(n) &= -a_{B/C}x(n) + d(1-a_{B/C})x(n-1) + x(n-2) \\ &\quad -d(1-a_{B/C})y_1(n-1) + a_{B/C}y_1(n-2) \end{aligned} \quad (2.56)$$

$$y(n) = \frac{H_0}{2} [x(n) - y_1(n)] + x(n). \quad (2.57)$$

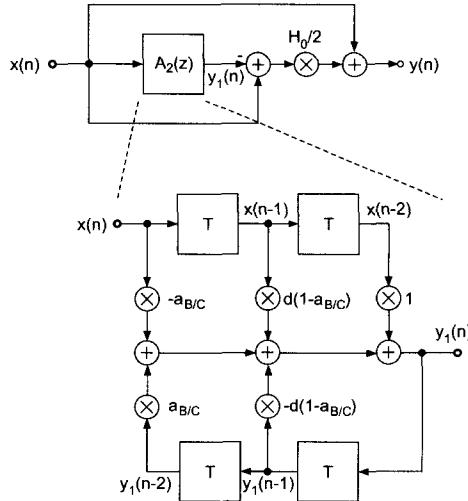


Figure 2.22 Second-order peak filter.

The center/cut-off frequency parameter d and the coefficient H_0 are given by

$$d = -\cos(2\pi f_c/f_s) \quad (2.58)$$

$$V_0 = H(e^{j2\pi f_c/f_s}) = 10^{G/20} \quad (2.59)$$

$$H_0 = V_0 - 1. \quad (2.60)$$

The bandwidth f_b is adjusted through the parameters a_B and a_C for boost and cut and are given by

$$a_B = \frac{\tan(\pi f_b/f_s) - 1}{\tan(\pi f_b/f_s) + 1} \quad (2.61)$$

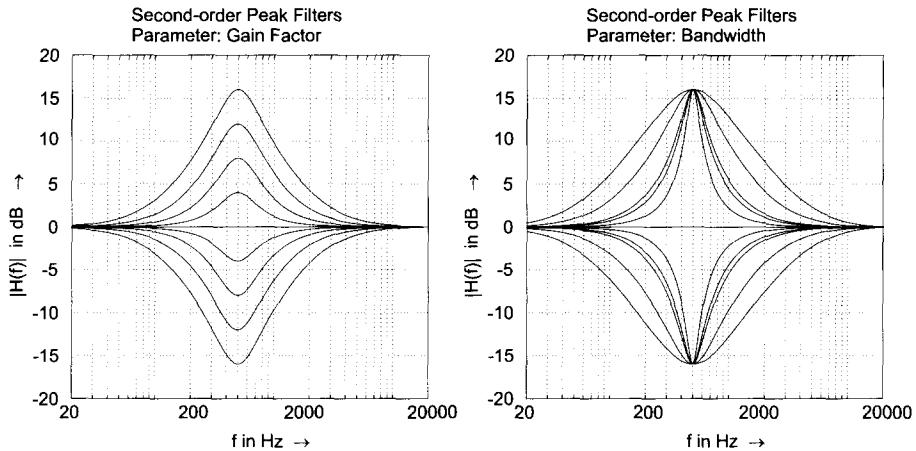
$$a_C = \frac{\tan(\pi f_b/f_s) - V_0}{\tan(\pi f_b/f_s) + V_0}. \quad (2.62)$$

This peak filter offers almost independent control of all three musical parameters center/cut-off frequency, bandwidth and gain. Another design approach from [Zöl97] shown in Table 2.4 allows direct computation of the five coefficients for a second-order transfer function as given in the difference equation (2.2).

Frequency responses for several settings of a peak filter are shown in Fig. 2.23. The left part shows a variation of the gain with a fixed center frequency and bandwidth. The right part show for fixed gain and center frequency a variation of the bandwidth or Q factor.

Table 2.4 Peak filter design with $K = \tan(\pi f_c/f_s)$ [Zöl97].

peak (boost $V_0 = 10^{G/20}$)				
b_0	b_1	b_2	a_1	a_2
$\frac{1 + \frac{V_0}{Q_\infty} K + K^2}{1 + \frac{1}{Q_\infty} K + K^2}$	$\frac{2(K^2 - 1)}{1 + \frac{1}{Q_\infty} K + K^2}$	$\frac{1 - \frac{V_0}{Q_\infty} K + K^2}{1 + \frac{1}{Q_\infty} K + K^2}$	$\frac{2(K^2 - 1)}{1 + \frac{1}{Q_\infty} K + K^2}$	$\frac{1 - \frac{1}{Q_\infty} K + K^2}{1 + \frac{1}{Q_\infty} K + K^2}$
peak (cut $V_0 = 10^{-G/20}$)				
b_0	b_1	b_2	a_1	a_2
$\frac{1 + \frac{1}{Q_\infty} K + K^2}{1 + \frac{V_0}{Q_\infty} K + K^2}$	$\frac{2(K^2 - 1)}{1 + \frac{V_0}{Q_\infty} K + K^2}$	$\frac{1 - \frac{1}{Q_\infty} K + K^2}{1 + \frac{V_0}{Q_\infty} K + K^2}$	$\frac{2(K^2 - 1)}{1 + \frac{V_0}{Q_\infty} K + K^2}$	$\frac{1 - \frac{V_0}{Q_\infty} K + K^2}{1 + \frac{V_0}{Q_\infty} K + K^2}$

**Figure 2.23** Frequency responses second-order peak filters.

2.4 Time-varying Filters

The parametric filters discussed in the previous sections allow the time-varying control of the filter parameters gain, cut-off frequency and bandwidth or Q factor. Special applications of time-varying audio filters will be shown in the following.

2.4.1 Wah-wah Filter

The wah-wah effect is produced mostly by foot-controlled signal processors containing a bandpass filter with variable center/resonant frequency and a small bandwidth. Moving the pedal back and forth changes the bandpass cut-off/center frequency. The “wah-wah” effect is then mixed with the direct signal as shown in Fig. 2.24. This effect leads to a spectrum shaping similar to speech and produces a speech like “wah-wah” sound. If the variation of the center frequency is controlled by the

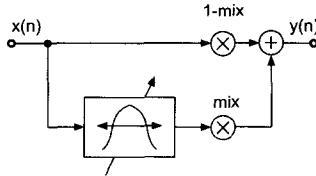


Figure 2.24 Wah-wah: time-varying bandpass filter.

input signal, a low-frequency oscillator is used to change the center frequency. Such an effect is called an auto-wah filter. If the effect is combined with a low-frequency amplitude variation, which produces a tremolo, the effect is denoted a tremolo-wah filter. Replacing the unit delay in the bandpass filter by an M tap delay leads to the M -fold wah-wah filter [Dis99], which is shown in Fig. 2.25. M bandpass filters are spread over the entire spectrum and simultaneously change their center frequency. When a white noise input signal is applied to an M -fold wah-wah filter, a spectrogram of the output signal shown in Fig. 2.26 illustrates the periodic enhancement of the output spectrum. Table 2.5 contains several parameter settings for different effects.

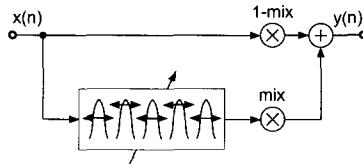


Figure 2.25 M -fold wah-wah filter.

Table 2.5 Effects with M -fold wah-wah filter [Dis99].

	M	Q^{-1}/f_m	$\Delta\omega/2\pi$
Wah-Wah	1	-/3kHz	200Hz
M -fold Wah-Wah	5-20	0.5/-	200-500Hz
Bell effect	100	0.5/-	100Hz

2.4.2 Phaser

The previous effect relies on varying the center frequency of a bandpass filter. Another effect uses notch filters: *phasing*. A set of notch filters, that can be realized as a cascade of second-order IIR sections, is used to process the input signal. The output of the notch filters is then combined with the direct sound. The frequencies of the notches are slowly varied using a low-frequency oscillator (Figure 2.27) [Smi84]. “The strong phase shifts that exist around the notch frequencies combine

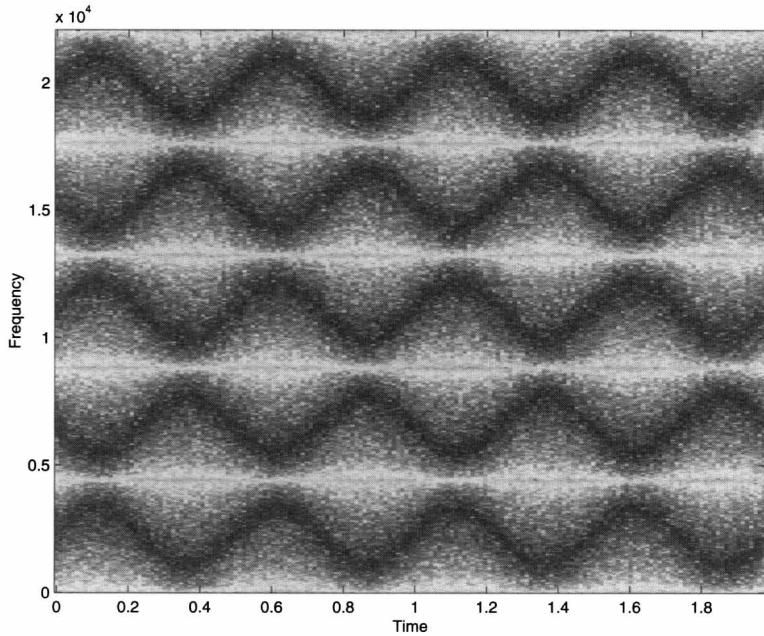


Figure 2.26 Spectrogram of output signal of a time-varying M-fold wah-wah filter [Dis99].

with the phases of the direct signal and cause phase cancellations or enhancements that sweep up and down the frequency axis" [Orf96]. Although this effect does not rely on a delay line, it is often considered to go along with delay-line based effects because the sound effect is similar to that of *flanging*. An extensive discussion on this topic is found in [Str83]. A different phasing approach is shown in Figure 2.28. The notch filters have been replaced by second-order allpass filters with time-varying center frequencies. The cascade of allpass filters produces time-varying phase shifts which lead to cancellations and amplifications of different frequency bands when used in the feedforward and feedback configuration.

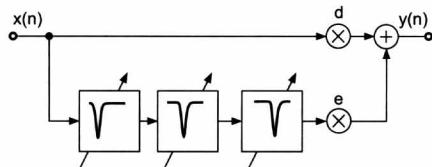


Figure 2.27 Phasing.

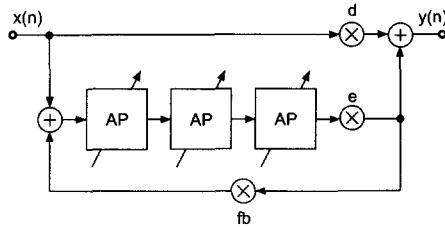


Figure 2.28 Phasing with time-varying allpass filters.

2.4.3 Time-varying Equalizers

- Time-varying octave bandpass filters, as shown in Fig. 2.29, offer the possibility of achieving wah-wah-like effects. The spectrogram of the output signal in Fig. 2.30 demonstrates the octave spaced enhancement of this approach.

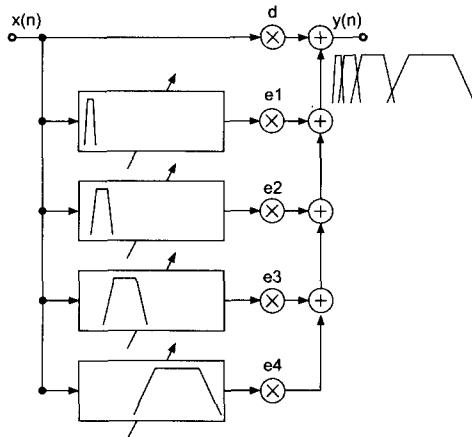


Figure 2.29 Time-varying octave filters.

- Time-varying shelving and peak filters: the special allpass realization of shelving and peak filters has shown that a combination of lowpass, bandpass and allpass filters gives access to several frequency bands inside such a filter structure. Integrating level measurement or envelope followers (see Chapter 5) into these frequency bands can be used for adaptively changing the filter parameters gain, cut-off/center frequency and bandwidth or Q factor. The combination of dynamics processing, which will be discussed in Chapter 5, and parametric filter structures allows the creation of signal dependent filtering effects with a variety of applications.

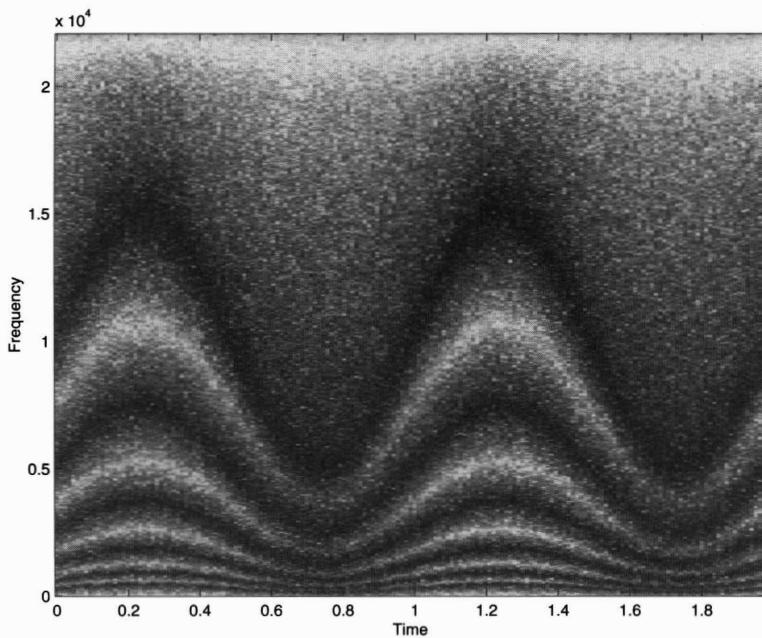


Figure 2.30 Spectrogram of output signal for time-varying octave filters.

- Feedback cancellers, which are based on time-varying notch filters, play an important role in sound reinforcement systems. The spectrum is continuously monitored for spectral peaks and a very narrow-band notch filter is applied to the signal path.

2.5 Conclusion

Filtering is still one of the most commonly used effect tools for sound recording and production. Nevertheless, its successful application is heavily dependent on the specialized skills of the operator. In this chapter we have described basic filter algorithms for time-domain audio processing. These algorithms perform the filtering operations by the computation of difference equations. The coefficients for the difference equations are given for several filter functions such as lowpass, highpass, bandpass, shelving and peak filters. Simple design formulas for various equalizers lead to efficient implementations for time-varying filter applications. The combination of these basic filters together with the signal processing algorithms of the following chapters allows the building of more sophisticated effects.

Sound and Music

- [m-chu5sna] chu5sna: vocoder speech convolved with snare-drum strokes. Demo Sound. DAFX Sound Library.
- [m-gendsna] gendsna: snare-drum rhythm pattern is mapped onto a gender sound. Demo Sound. DAFX Sound Library.
- [m-Mai97] M. Maiguashca: Reading Castañeda. CD. Wergo2053-2, zkm 3 edition, 1997.
- [m-Pie99] F. Pieper: Das Effekte Praxisbuch. GC Carstensen, 1999. CD, Tr. 1, 35, 36.
- [m-quasthal] quasthal: convolution of speech with impulse responses. Demo Sound. DAFX Sound Library.
- [m-Vag96] H. Vaggione: MYR-S, Composition for cello, electroacoustic set-up and tape. Festival Synthèse, Bourges 1996.
- [m-Vas93] P. Vasseur: PURPLE FRAME, in Le sens caché 100 mystères, CD FREE WAY MUSIQUE, No. 869193, 1993.
- [m-tubg5sna] tubg5sna: tuba glissando convolved by a series of snare-drum strokes. Demo Sound. DAFX Sound Library.

Bibliography

- [Bri94] R. Bristow. The equivalence of various methods of computing biquad coefficients for audio parametric equalizers. In *Proc. 97th AES Convention*, Preprint 3906, San Francisco, 1994.
- [Cha80] H. Chamberlin. *Musical Applications of Microprocessors*. Hayden Book Company, 1980.
- [Dat97] J. Dattoro. Effect design, part 1: Reverberator and other filters. *J. Audio Eng. Soc.*, 45(9):660–684, September 1997.
- [Die00] S. Diedrichsen. Personal communication. emagic GmbH, 2000.
- [Dis99] S. Disch. Digital audio effects — modulation and delay lines. Master's thesis, Technical University Hamburg-Harburg, 1999.
- [DJ85] C. Dodge and T. A. Jerse. *Computer Music; Synthesis, Composition and Performance*. Schirmer Books, 1985.

- [DMT99] P. Dutilleux and C. Müller-Tomfelde. AML | Architecture and Music Laboratory, a museum installation. In *Proc. of the 16th AES Int. Conf. on Spatial Sound Reproduction. Rovaniemi, Finland. Audio Engineering Society*, pp. 191–206, April 10–12 1999.
- [Dut89a] P. Dutilleux. Simple to operate digital time-varying filters. In *Proc. 86th AES Convention*, Preprint 2757, Hamburg, March 1989.
- [Dut89b] P. Dutilleux. Spinning the sounds in real-time. In *Proc. International Computer Music Conference*, pp. 94–97, November Columbus 1989.
- [Dut91] P. Dutilleux. *Vers la machine à sculpter le son, modification en temps réel des caractéristiques fréquentielles et temporelles des sons*. PhD thesis, University of Aix-Marseille II, 1991.
- [Gar95] W.G. Gardner. Efficient convolution without input-output delay. *J. Audio Eng. Soc.*, 43(3):127–136, March 1995.
- [HB93] F. Harris and E. Brooking. A versatile parametric filter using an imbeded all-pass sub-filter to independently adjust bandwidth, center frequency, and boost or cut. In *95th AES Convention*, Preprint 3757, New York, 1993.
- [LKG72] M. Labarrère, J. Krief, and B. Gimonet. Le filtrage et ses applications. *CEPADUES*, 1972.
- [Mai97] M. Maiguashca. *Reading Castañeda. Booklet*. Wergo2053-2, zkm 3 edition, 1997.
- [McG93] D.S. McGrath. An efficient 30-band graphic equalizer implementation for a low cost dsp processor. In *Proc. 95th AES Convention*, Preprint 3756, New York, 1993.
- [MT99] C. Müller-Tomfelde. Low-latency convolution for real-time applications. In *Proc. of the 16th AES Int. Conf. on Spatial Sound Reproduction, Rovaniemi, Finland*, pp. 454–460. Audio Engineering Society, April 10–12 1999.
- [Orf96] S.J. Orfanidis. *Introduction to Signal Processing*. Prentice-Hall, 1996.
- [Orf97] S.J. Orfanidis. Digital parametric equalizer design with prescribed nyquist-frequency gain. *J. Audio Eng. Soc.*, 45(6):444–455, June 1997.
- [RM87] P.A. Regalia and S.K. Mitra. Tunable digital frequency response equalization filters. *IEEE Trans. Acoustics, Speech and Signal Processing*, 35(1):118–120, January 1987.
- [Roa97] C. Roads. Sound transformation by convolution. In C. Roads, St. Pope, A. Piccialli, and G. De Poli (eds), *Musical Signal Processing*, pp. 411–438. Swets & Zeitlinger Publishers, 1997.

- [Smi84] J.O. Smith. An all-pass approach to digital phasing and flanging. In *Proc. International Computer Music Conference*, pp. 103–109, 1984.
- [Str83] A. Strange. *Electronic Music, Systems, Techniques and Controls*. W. C. Brown, 1983.
- [Vag98] H. Vaggione. Transformations morphologiques: quelques exemples. In *Proceedings of JIM98, CNRS-LMA*, pp. G1.1–G1.10, Marseille 1998.
- [Whi86] S.A. White. Design of a digital biquadratic peaking or notch filter for digital audio equalization. *J. Audio Eng. Soc.*, 34(6):479–482, June 1986.
- [Zöl97] U. Zölzer. *Digital Audio Signal Processing*. John Wiley & Sons, Ltd, 1997.

Chapter 3

Delays

P. Dutilleux, U. Zölzer

3.1 Introduction

Delays can be experienced in acoustical spaces. A sound wave reflected by a wall will be superimposed on the sound wave at the source. If the wall is far away, such as a cliff, we will hear an echo. If the wall is close to us, we will notice the reflections through a modification of the sound color. Repeated reflections can appear between parallel boundaries. In a room, such reflections will be called *flatter echo*. The distance between the boundaries determines the delay that is imposed to each reflected sound wave. In a cylinder, successive reflections will develop at both ends. If the cylinder is long, we will hear an iterative pattern whereas, if the cylinder is short, we will hear a pitched tone. Equivalents of these acoustical phenomena have been implemented as signal processing units.

3.2 Basic Delay Structures

3.2.1 FIR Comb Filter

The network that simulates a single delay is called the FIR comb filter. The input signal is delayed by a given time duration. The effect will be audible only when the processed signal is combined (added) to the input signal, which acts here as a reference. This effect has 2 tuning parameters: the amount of time delay τ and the relative amplitude of the delayed signal to that of the reference signal. The difference equation and the transfer function are given by

$$y(n) = x(n) + gx(n - M) \quad (3.1)$$

$$\text{with } M = \tau/f_s \quad (3.2)$$

$$H(z) = 1 + gz^{-M}. \quad (3.3)$$

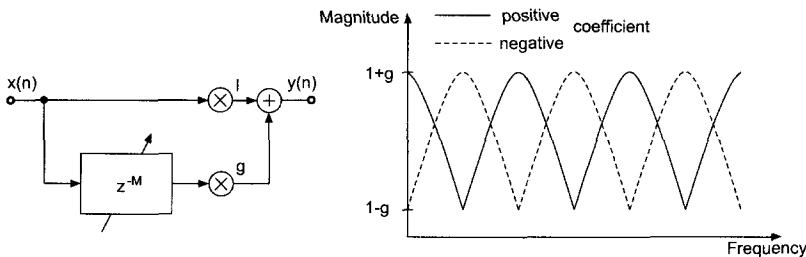


Figure 3.1 FIR comb filter and magnitude response.

The time response of this filter is made up of the direct signal and the delayed version. This simple time domain behavior comes along with interesting frequency domain patterns. For positive values of g , the filter amplifies all frequencies that are multiples of $1/\tau$ and attenuates all frequencies that lie in between. The transfer function of such a filter shows a series of spikes and it looks like a comb (Fig. 3.1). That is why this type of filter is called a comb filter. For negative values of g , the filter attenuates frequencies that are multiples of $1/\tau$ and amplifies those that lie in between. The gain varies between $1 + g$ and $1 - g$ [Orf96]. The following M-file 3.1 demonstrates a sample-by-sample based FIR comb filter. For plotting the output signal use the command `stem(0:length(y)-1,y)` and for the evaluation of the magnitude and phase response use the command `freqz(y,1)`.

M-file 3.1 (fircomb.m)

```
x=zeros(100,1);x(1)=1; % unit impulse signal of length 100
g=0.5;
Delayline=zeros(10,1);% memory allocation for length 10
for n=1:length(x);
y(n)=x(n)+g*Delayline(10);
Delayline=[x(n);Delayline(1:10-1)];
end;
```

As well as acoustical delays, the FIR comb filter has an effect both in the time and frequency domains. Our ear is more sensitive to the one aspect or to the other according to the range where the time delay is set. For larger values of τ , we can hear an echo that is distinct from the direct signal. The frequencies that are amplified by the comb are so close to each other that we barely identify the spectral effect. For smaller values of τ , our ear can no longer segregate the time events but can notice the spectral effect of the comb.

3.2.2 IIR Comb Filter

Similar to the endless reflections at both ends of a cylinder, the IIR comb filter produces an endless series of responses $y(n)$ to an input $x(n)$. The input signal

circulates in a delay line that is fed back to the input. Each time the signal goes through the delay line it is attenuated by g . It is sometimes necessary to scale the input signal by c in order to compensate for the high amplification produced by the structure. It is implemented by the structure shown in Fig. 3.2 with the difference equation and transfer function given by

$$y(n) = cx(n) + gy(n - M), \text{ with } M = \tau / f_s \quad (3.4)$$

$$H(z) = c/(1 - gz^{-M}). \quad (3.5)$$

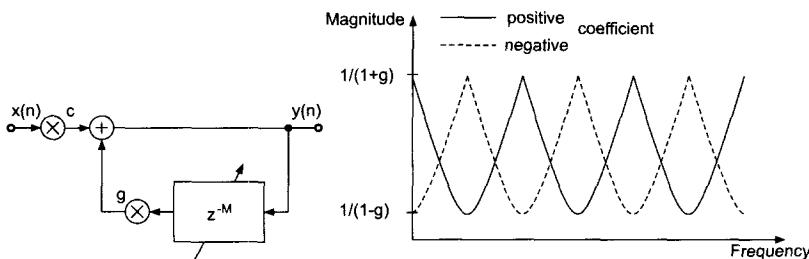


Figure 3.2 IIR comb filter and magnitude response.

Due to the feedback loop, the time response of the filter is infinite. After each time delay τ a copy of the input signal will come out with an amplitude g^p where p is the number of cycles that the signal has gone through the delay line. It can readily be seen, that $|g| \leq 1$ is a stability condition. Otherwise the signal would grow endlessly. The frequencies that are affected by the IIR comb filter are similar to those affected by the FIR comb filter. The gain varies between $1/(1 - g)$ and $1/(1 + g)$. The main differences between the IIR comb and the FIR comb is that the gain grows very high and that the frequency peaks get narrower as $|g|$ comes closer to 1 (see Fig. 3.2). The following M-file 3.2 shows the implementation of a sample-by-sample based IIR comb filter.

M-file 3.2 (iircmb.m)

```
x=zeros(100,1);x(1)=1; % unit impulse signal of length 100
g=0.5;
Delayline=zeros(10,1); % memory allocation for length 10
for n=1:length(x);
y(n)=x(n)+g*Delayline(10);
Delayline=[y(n);Delayline(1:10-1)];
end;
```

3.2.3 Universal Comb Filter

The combination of FIR and IIR comb filters leads to the universal comb filter. This is simply an allpass filter (see Fig. 2.8) where the one sample delay operator

z^{-1} is replaced by the M sample delay operator z^{-M} and an additional multiplier FF shown in Figure 3.3. The special cases for differences in feedback parameter FB, feedforward parameter FF and blend parameter BL are given in Table 3.1. M-file 3.3 shows the implementation of a sample-by-sample universal comb filter.

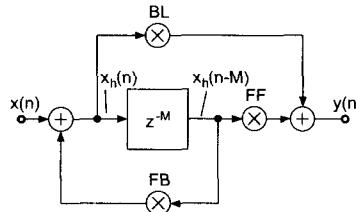


Figure 3.3 Universal comb filter.

Table 3.1 Parameters for universal comb filter.

	BL	FB	FF
FIR comb filter	X	0	X
IIR comb filter	1	X	0
allpass	a	$-a$	1
delay	0	0	1

M-file 3.3 (unicomb.m)

```

x=zeros(100,1);x(1)=1; % unit impulse signal of length 100
BL=0.5;
FB=-0.5;
FF=1;
M=10;
Delayline=zeros(M,1); % memory allocation for length 10
for n=1:length(x);
    xh=x(n)+FB*Delayline(M);
    y(n)=FF*Delayline(M)+BL*xh;
    Delayline=[xh;Delayline(1:M-1)];
end;

```

The extension of the above universal comb filter to a parallel connection of N comb filters is shown in Figure 3.4. The feedback, feedforward and blend coefficients are now $N \times N$ matrices to mix the input and output signals of the delay network. The use of different parameter sets leads to the applications shown in Table 3.2.

3.2.4 Fractional Delay Lines

Variable-length delays of the input signal are used to simulate several acoustical effects. Therefore, delays of the input signal with noninteger values of the sampling

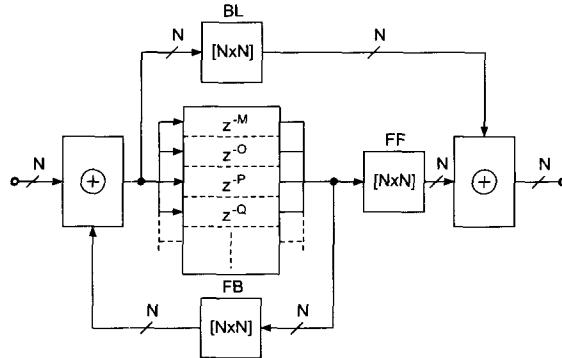


Figure 3.4 Generalized structure of parallel allpass comb filters.

Table 3.2 Effects with generalized comb filter.

	delay	BL	FB	FF
slapback	50ms	1	0	X
echo	>50ms	1	$0 < X < 1$	0
reverb		matrix	matrix	matrix

interval are necessary. A delay of the input signal by M samples plus a fraction of the normalized sampling interval with $0 \leq \text{frac} \leq 1$ is given by

$$y(n) = x(n - [M + \text{frac}]) \quad (3.6)$$

and can be implemented by a fractional delay shown in Fig. 3.5.

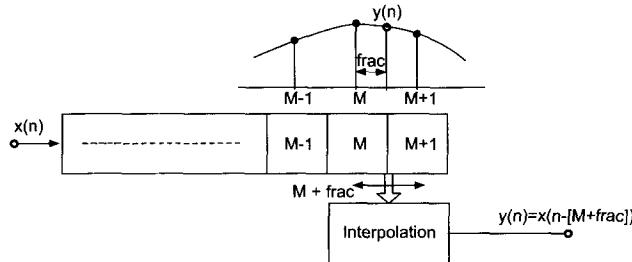


Figure 3.5 Fractional delay line with interpolation.

Design tools for fractional delay filters can be found in [LVKL96]. An interpolation algorithm has to compute the output sample $y(n)$, which lies in between the two samples at time instants M and $M + 1$. Several interpolation algorithms have been proposed for audio applications:

- linear interpolation [Dat97]

$$y(n) = x(n - [M + 1])\text{frac} + x(n - M)(1 - \text{frac}) \quad (3.7)$$

- allpass interpolation [Dat97]

$$y(n) = x(n - [M + 1])\text{frac} + x(n - M)(1 - \text{frac}) - y(n - 1)(1 - \text{frac}) \quad (3.8)$$

- sinc interpolation [FC98]

- fractionally addressed delay lines [Roc98, Roc00]

- spline interpolation [Dis99]

$$\begin{aligned} y(n) &= x(n - [M + 1]) \cdot \frac{\text{frac}^3}{6} \\ &+ x(n - M) \cdot \frac{(1 + \text{frac})^3 - 4 \cdot \text{frac}^3}{6} \\ &+ x(n - [M - 1]) \cdot \frac{(2 - \text{frac})^3 - 4(1 - \text{frac})^3}{6} \\ &+ x(n - [M - 2]) \cdot \frac{(1 - \text{frac})^3}{6}. \end{aligned} \quad (3.9)$$

They all perform interpolation of a fractional delayed output signal with different computational complexity and different performance properties, which are discussed in [Roc00]. The choice of the algorithm depends on the specific application.

3.3 Delay-based Audio Effects

3.3.1 Vibrato

When a car is passing by, we hear a pitch deviation due to the doppler effect [Dut91]. This effect will be explained in another chapter but we can keep in mind that the pitch variation is due to the fact that the distance between the source and our ears is being varied. Varying the distance is, for our application, equivalent to varying the time delay. If we keep on varying periodically the time delay we will produce a periodical pitch variation. This is precisely a vibrato effect. For that purpose we need a delay line and a low-frequency oscillator to drive the delay time parameter. We should only listen to the delayed signal. Typical values of the parameters are 5 to 10 ms as average delay-time and 5 to 14 Hz rate for the low-frequency oscillator (Figure 3.6) [And95, Whi93]. M-file 3.4 shows the implementation for vibrato [Dis99].

```
M-file 3.4 (vibrato.m)
% Vibrato
function y=vibrato(y,SAMPLERATE,Modfreq,Width)
ya_alt=0;
```

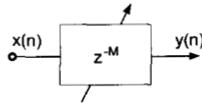


Figure 3.6 Vibrato.

```

Delay=Width; % basic delay of input sample in sec
DELAY=round(Delay*SAMPLERATE); % basic delay in # samples
WIDTH=round(Width*SAMPLERATE); % modulation width in # samples
if WIDTH>DELAY
    error('delay greater than basic delay !!!');
    return;
end
MODFREQ=Modfreq/SAMPLERATE; % modulation frequency in # samples
LEN=length(x); % # of samples in WAV-file
L=2+DELAY+WIDTH*2; % length of the entire delay
Delayline=zeros(L,1); % memory allocation for delay
y=zeros(size(x)); % memory allocation for output vector
for n=1:(LEN-1)
    M=MODFREQ;
    MOD=sin(M*2*pi*n);
    ZEIGER=1+DELAY+WIDTH*MOD;
    i=floor(ZEIGER);
    frac=ZEIGER-i;
    Delayline=[x(n);Delayline(1:L-1)];
    %---Linear Interpolation-----
    y(n,1)=Delayline(i+1)*frac+Delayline(i)*(1-frac);
    %---Allpass Interpolation-----
    %y(n,1)=(Delayline(i+1)+(1-frac)*Delayline(i)-(1-frac)*ya_alt);
    %ya_alt=ya(n,1);
    %---Spline Interpolation-----
    %y(n,1)=Delayline(i+1)*frac^3/6
    %....+Delayline(i)*((1+frac)^3-4*frac^3)/6
    %....+Delayline(i-1)*((2-frac)^3-4*(1-frac)^3)/6
    %....+Delayline(i-2)*(1-frac)^3/6;
    %3rd-order Spline Interpolation
end

```

3.3.2 Flanger, Chorus, Slapback, Echo

A few popular effects can be realized using the comb filter. They have special names because of the peculiar sound effects that they produce. Consider the FIR comb filter. If the delay is in the range 10 to 25 ms, we will hear a quick repetition named *slapback* or *doubling*. If the delay is greater than 50 ms we will hear an *echo*. If

the time delay is short (less than 15 ms) and if this delay time is continuously varied with a low frequency such as 1 Hz, we will hear the *flanging* effect. If several copies of the input signal are delayed in the range 10 to 25 ms with small and random variations in the delay times, we will hear the *chorus* effect, which is a combination of the vibrato effect with the direct signal (see Table 3.3 and Fig. 3.7) [Orf96, And95, Dat97]. These effects can also be implemented as IIR comb filters. The feedback will then enhance the effect and produce repeated slapbacks or echoes.

Table 3.3 Typical delay-based effects.

Delay range (ms) (Typ.)	Modulation (Typ.)	Effect name
0 ... 20	-	Resonator
0 ... 15	Sinusoidal	Flanging
10 ... 25	Random	Chorus
25 ... 50	-	Slapback
> 50	-	Echo

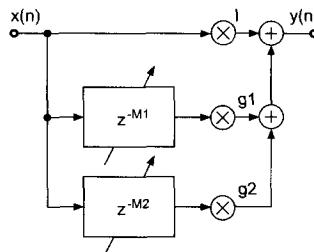


Figure 3.7 Chorus.

Normalization. We saw in 2.2.1 that it is important to compensate for the intrinsic gain of the filter structure. Whereas in practice the FIR comb filter does not amplify the signal by more than 6 dB, the IIR comb filter can yield a very large amplification when $|g|$ comes close to 1. The L_2 and L_∞ norm are given by

$$L_2 = 1/\sqrt{1 - g^2} \quad L_\infty = 1/(1 - |g|). \quad (3.10)$$

The normalization coefficient $c = 1/L_\infty$, when applied, ensures that no overload will occur with, for example, periodical input signals. $c = 1/L_2$ ensures that the loudness will remain approximatively constant for broadband signals.

A standard effect structure was proposed by Dattorro [Dat97] and is shown in Fig. 3.8. It is based on the allpass filter modification towards a general allpass comb, where the fixed delay line is replaced by a variable-length delay line. Dattorro [Dat97] proposed keeping the feedback tap of the delay line fixed, that means the input signal to the delay line $x_h(n)$ is delayed by a fixed integer delay K and with this $x_h(n - K)$ is weighted and fed back. The delay K is the center tab delay of

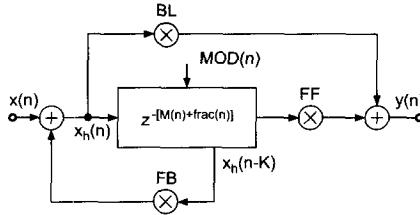


Figure 3.8 Standard effects with variable-length delay line.

the variable-length delay line for the feed forward path. The control signal $MOD(n)$ for changing the length of the delay line can either be a sinusoid or lowpass noise. Typical settings of the parameters are given in Table 3.4.

Table 3.4 Industry standard audio effects. [Dis99]

	BL	FF	FB	DELAY	DEPTH	MOD
Vibrato	0	1	0	0 ms	0-3 ms	0.1-5 Hz Sine
Flanger	0.7	0.7	0.7	0 ms	0-2 ms	0.1-1 Hz Sine
(white) Chorus	0.7	1	(-0.7)	1-30 ms	1-30 ms	lowpass noise
Doubling	0.7	0.7	0	10-100 ms	1-100 ms	lowpass noise

3.3.3 Multiband Effects

New interesting sounds can be achieved after splitting the signal into several frequency bands, for example, lowpass, bandpass and highpass signals, as shown in Fig. 3.9.

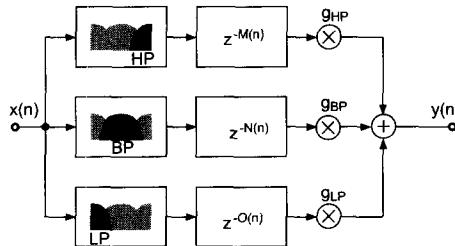


Figure 3.9 Multiband effects.

Variable-length delays are applied to these signals with individual parameter settings and the output signals are weighted and summed to the broad-band signal [FC98, Dis99]. Efficient frequency splitting schemes are available from loudspeaker crossover designs and can be applied for this purpose directly. One of these techniques uses complementary filtering [Fli94, Orf96], which consists of lowpass filtering

and subtracting the lowpass signal from the broad-band signal to derive the high-pass signal, as shown in Fig. 3.10. The lowpass signal is then further processed by a following stage of the same complementary technique to deliver the bandpass and lowpass signal.

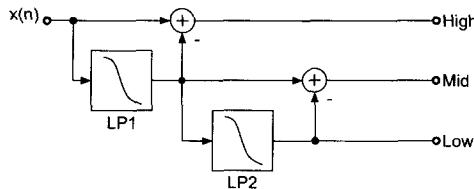


Figure 3.10 Filter bank for multiband effects.

3.3.4 Natural Sounding Comb Filter

We have made the comparison between acoustical cylinders and IIR comb filters. This comparison might seem inappropriate because the comb filters sound *metallic*. They tend to amplify greatly the high frequency components and they appear to resonate much too long compared to the acoustical cylinder. To find an explanation, let us consider the boundaries of the cylinder. They reflect the acoustic waves with an amplitude that decreases with frequency. If the comb filter should sound like an acoustical cylinder, then it should also have a frequency-dependent feedback coefficient $g(f)$. This frequency dependence can be realized by using a first order lowpass filter in the feedback loop (see Fig. 3.11) [Moo85].

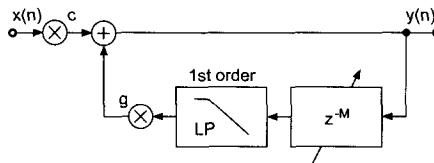


Figure 3.11 Lowpass IIR comb filter.

These filters sound more *natural* than the plain IIR comb filters. They find application in room simulators. Further refinements such as fractional delays and compensation of the frequency-dependent group-delay within the lowpass filter make them suitable for the imitation of acoustical resonators. They have been used for example to impose a definite pitch onto broadband signals such as sea waves or to detune fixed-pitched instruments such as a Celtic harp [m-Ris92]. M-file 3.5 shows the implementation for a sample-by-sample based lowpass IIR comb filter.

M-file 3.5 (lpiircomb.m)

```
x=zeros(100,1);x(1)=1; % unit impulse signal of length 100
g=0.5;
```

```

b_0=0.5;
b_1=0.5;
a_1=0.7;
xhold=0;yhold=0;
Delayline=zeros(10,1); % memory allocation for length 10
for n=1:length(x);
    yh(n)=b_0*Delayline(10)+b_1*xhold-a_1*yhold;
    % 1st-order difference equation
    yhold=yh(n);
    xhold=Delayline(10);
    y(n)=x(n)+g*yh(n);
    Delayline=[y(n);Delayline(1:10-1)];
end;

```

3.4 Conclusion

Delays are used in audio processing to solve several practical problems, for example delay compensation for sound reinforcement systems, and as basic building blocks for delay-based audio effects, artificial reverberation and physical models for instrument simulation. The variety of applications of delays to *spatial effects* will be presented in Chapter 6.

This brief introduction has described some of the basic delay structures, which should enable the reader to implement and experiment with delay algorithms on their own. We have focused on a small set of important delay applications such as echo, vibrato, flanger and chorus. We have pointed out the important combination of delays and filters and their time-varying application. These basic building blocks may serve as a source of ideas for designing new digital audio effects.

Sound and Music

- [m-Ris92] J.-C. Risset: Lurai, pour harpe celtique et bande. Radio France, 21.3.1992.
- [m-Pie99] F. Pieper: Das Effekte Praxisbuch. Delay, Kammfilter, Phaser, Vibrato, Chorus. CD, Tr. 2-7, 18-24, 28-32, Ch. 5, 7, 8, 10, 11. GC Carstensen, 1999.

Bibliography

- [And95] C. Anderton. *Multieffects for Musicians*. Amsco Publications, 1995.
- [Dat97] J. Dattoro. Effect design, part 2: Delay-line modulation and chorus. *J. Audio Eng. Soc.*, 45(10):764–788, October 1997.

- [Dis99] S. Disch. Digital audio effects — modulation and delay lines. Master's thesis, Technical University Hamburg-Harburg, 1999.
- [Dut91] P. Dutilleux. *Vers la machine à sculpter le son, modification en temps réel des caractéristiques fréquentielles et temporelles des sons*. PhD thesis, University of Aix-Marseille II, 1991.
- [FC98] P. Fernández-Cid and F.J. Casajús-Quirós. Enhanced quality and variety of chorus/flange units. In *Proc. DAFX-98 Digital Audio Effects Workshop*, pp. 35–39, Barcelona, November 1998.
- [Fli94] N.J. Fliege. *Multirate Digital Signal Processing*. John Wiley & Sons, Ltd, 1994.
- [LVKL96] T.I. Laakso, V. Välimälki, M. Karjalainen, and U.K. Laine. Splitting the unit delay. *IEEE Signal Processing Magazine*, 13:30–60, 1996.
- [Moo85] J.A. Moorer. About this reverberation business. In *Foundations of Computer Music*, pp. 605–639. MIT Press, 1985.
- [Orf96] S.J. Orfanidis. *Introduction to Signal Processing*. Prentice-Hall, 1996.
- [Roc98] D. Rochesso. Fractionally-adressed delay lines. In *Proc. DAFX-98 Digital Audio Effects Workshop*, pp. 40–43, Barcelona, November 1998.
- [Roc00] D. Rochesso. Fractionally addressed delay lines. *IEEE Trans. on Speech and Audio Processing*, 8(6):717–727, November 2000.
- [Whi93] P. White. *L'enregistrement créatif, Effets et processeurs, Tomes 1 et 2*. Les cahiers de l'ACME, 1993.

Chapter 4

Modulators and Demodulators

P. Dutilleux, U. Zölzer

4.1 Introduction

Modulation is the process by which parameters of a sinusoidal signal (amplitude, frequency and phase) are modified or varied by an audio signal. In the realm of telecommunications the word *modulate* means: “to shift the frequency spectrum of a signal to another frequency band”. Numerous techniques have been designed to achieve this goal and some have found applications in digital audio effects. In the field of audio processing these modulation techniques are mainly used with very low frequency shifts of the audio signal. In particular, the variation of control parameters for filters or delay lines can be regarded as an amplitude modulation or phase modulation of the audio signal. Wah-wah, phaser and tremolo are typical examples of amplitude modulation and vibrato, flanger and chorus are examples for phase modulations of the audio signal. To gain a deeper understanding of the possibilities of modulation techniques we will first introduce simple schemes for amplitude modulation, single side band modulation and phase modulation and point out their use for audio effects. The combination of these modulators will lead to more advanced digital audio effects, which will be demonstrated by several examples. In a further section we will describe several demodulators, which extract the incoming signal or parameters of the incoming signal for further effects processing.

4.2 Modulators

4.2.1 Ring Modulator

In the *ring modulation* (RM) the audio signal $x(n)$ is multiplied by a sinusoid $m(n)$ with carrier frequency f_c . In the analog domain it was pretty difficult to do it properly but within a computer it is straightforward [Ste87] since it is a mere multiplication. The input signal is called the modulator $x(n)$ and the second operand is called the carrier $m(n)$:

$$y(n) = x(n) \cdot m(n). \quad (4.1)$$

If $m(n)$ is a sine wave of frequency f_c , the spectrum of the output $y(n)$ is made up of two copies of the input spectrum: the lower side band (LSB) and the upper side band (USB). The LSB is reversed in frequency and both sidebands are centered around f_c (Figure 4.2). Depending on the width of the spectrum of $x(n)$ and on the carrier frequency, the side bands can be partly mirrored around the origin of the frequency axis. If the carrier signal comprises several spectral components, the same effect happens with each component. Although the audible result of a ring modulation is fairly easy to comprehend for elementary signals, it gets very complicated with signals having numerous partials. The carrier itself is not audible in this kind of modulation. When carrier and modulator are sine waves of frequencies f_c and f_x , one hears the sum and the difference frequencies $f_c + f_x$ and $f_c - f_x$ [Hal95].

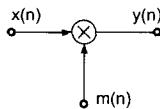


Figure 4.1 Ring modulation of a signal $x(n)$ by a sinusoidal carrier-signal $m(n)$.

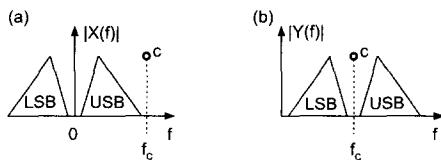


Figure 4.2 Ring modulation of a signal $x(n)$ by a sinusoidal carrier-signal $m(n)$. The spectrum of the modulator $x(n)$ (a) is shifted around the carrier frequency (b).

When the input signal is periodic with fundamental frequency f_0 , a sinusoidal carrier of frequency f_c produces a spectrum with amplitude lines at the frequencies $|k f_0 \pm f_c|$ [De 00]. A musical application of this effect is applied in the piece “Ofanim” by Luciano Berio. The first section is dominated by a duet between a child voice and a clarinet. The transformation of the child voice into a clarinet was desired. For this purpose a pitch detector computes the instantaneous frequency $f_0(n)$ of the

voice. Then the child voice passes through a ring modulator, where the frequency of the carrier f_c is set to $f_0(n)/2$. In this case odd harmonics prevail which is similar to the sound of a clarinet in the low register [Vid91]. Notice that in order to better represent the characteristic articulations of the clarinet and to eliminate typical vocal portamento, a sample-and-hold unit is inserted after the pitch detector for holding the current pitch until the successive one has been established.

4.2.2 Amplitude Modulator

The *amplitude modulation* (AM) was easier to realize with analog electronic means and has therefore been in use for a much longer time. It can be implemented by

$$y(n) = [1 + \alpha m(n)] \cdot x(n) \quad (4.2)$$

where it is assumed that the peak amplitude of $m(n)$ is 1. The α coefficient determines the depth of modulation. The modulation effect is maximum with $\alpha = 1$ and the effect is disengaged when $\alpha = 0$. A typical application is with an audio signal as carrier $x(n)$ and a *low-frequency oscillator* (LFO) as modulator $m(n)$ (Figure 4.3). The amplitude of the audio signal varies according to the amplitude of the LFO, and it is heard as such.

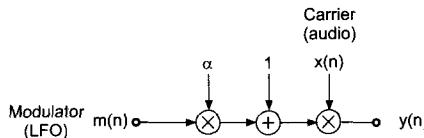


Figure 4.3 Typical application of AM.

When the modulator is an audible signal and the carrier a sine wave of frequency f_c , the spectrum of the output $y(t)$ is similar to that of the ring modulator except that the carrier frequency can be also heard. When carrier and modulator are sine waves of frequencies f_c and f_x , one hears three components: carrier, difference and sum frequencies ($f_c - f_x$, f_c , $f_c + f_x$). One should notice that due to the integration time of our hearing system the effect is perceived in a different manner depending on the frequency range of the signals. A modulation with frequencies below 20 Hertz will be heard in the time domain (variation of the amplitude, *tremolo* in Fig. 4.4) whereas modulations by high frequencies will be heard as distinct spectral components (LSB, carrier, USB).

4.2.3 Single-Side Band Modulator

The upper and lower side bands carry the same information although organized differently. In order to save bandwidth and transmitter power, radio-communication engineers have designed the single-side band (SSB) modulation scheme (Fig. 4.5). Either the LSB or the USB is transmitted. Phase shifted versions by 90° of the

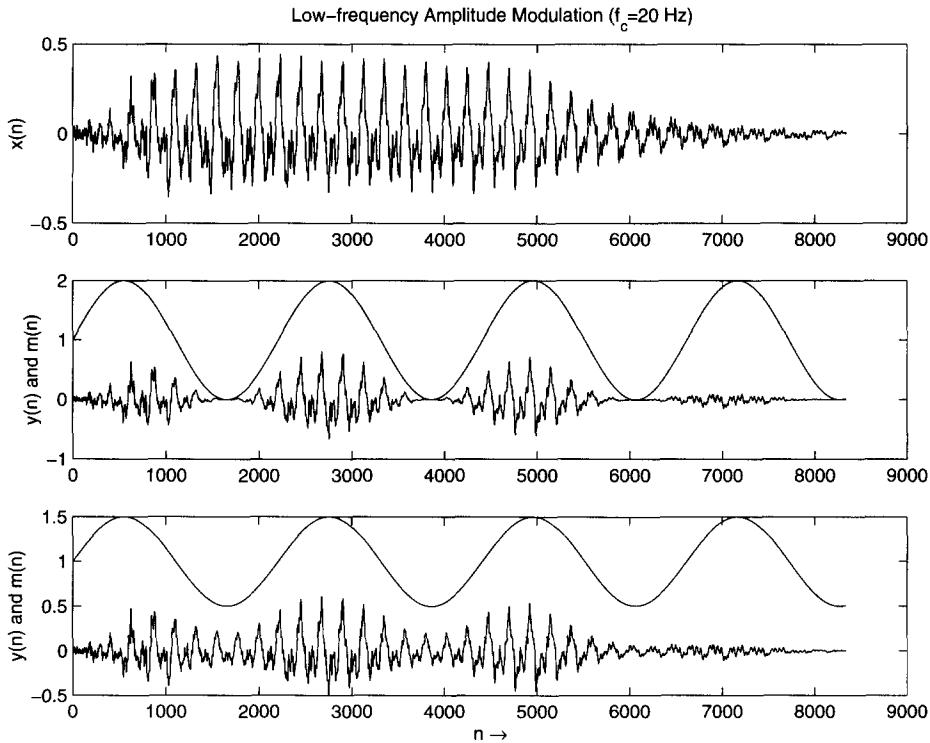


Figure 4.4 Tremolo effect by AM.

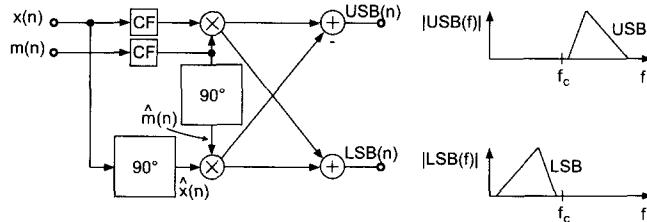


Figure 4.5 Single side band modulator with compensation filter CF and Hilbert filter (90° block).

modulating audio signal $x(n)$ are denoted by $\hat{x}(n)$ and of the carrier signal $m(n)$ by $\hat{m}(n)$ and are produced by Hilbert transform filters [Orf96]. The upper and lower side band signals can be computed as follows:

$$USB(n) = x(n)m(n) - \hat{x}(n)\hat{m}(n) \quad (4.3)$$

$$LSB(n) = x(n)m(n) + \hat{x}(n)\hat{m}(n). \quad (4.4)$$

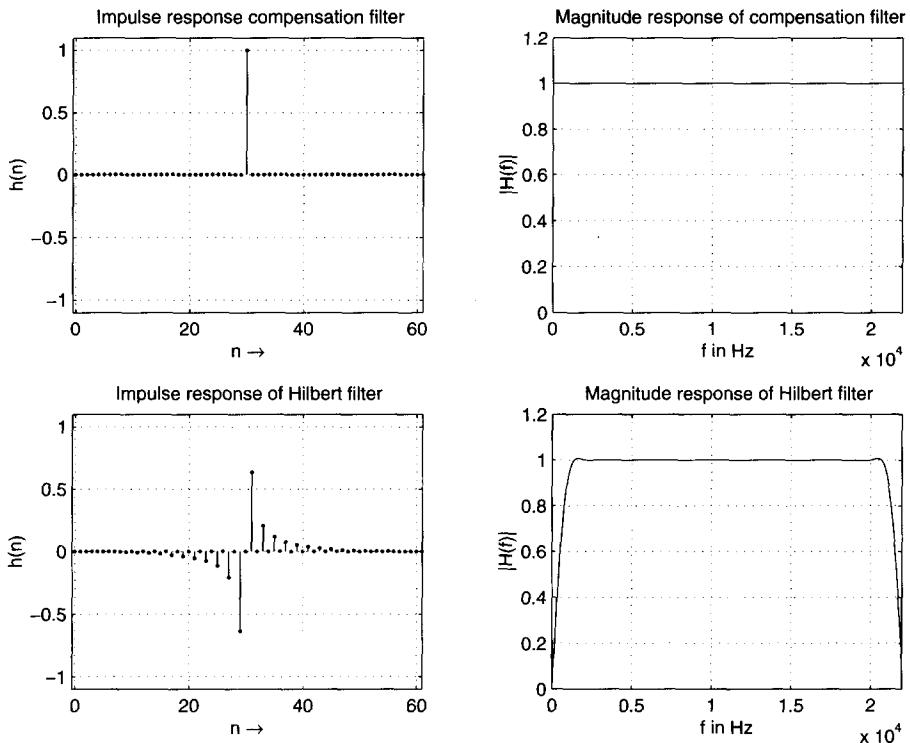


Figure 4.6 Delay compensation and Hilbert filter.

A discrete-time Hilbert transform can be approximated by a FIR filter with the zero-phase impulse response

$$a(n) = \frac{1 - \cos(\pi n)}{\pi n} = \begin{cases} 2/(\pi n) & \text{for } n \text{ odd} \\ 0 & \text{for } n \text{ even.} \end{cases} \quad (4.5)$$

These coefficients are multiplied with a suitable window function of length N , for example a Hamming window, and shifted right by $\frac{N-1}{2}$ to make the filter causal. Acceptable quality can be achieved with $N \approx 60$. Note that the use of the FIR Hilbert filter requires a delay in the direct path for the audio and the carrier signal. Figure 4.6 shows an example with the compensation delay of 30 samples and a FIR Hilbert filter of length $N = 61$. This effect is typically used with a sine wave as carrier of frequency f_c . The use of a complex oscillator for $m(n)$ simplifies the implementation. By using positive or negative frequencies it is then possible to select the USB or the LSB. The spectrum of $x(n)$ is frequency shifted up or down according to f_c . The results are non-harmonic sound effects: a plucked-string sound is heard, after processing, like a drum sound. The modification in pitch is much less than expected, probably because our ear recovers pitch information from the frequency difference between partials and not only from the lowest partial of the tone [Dut91].

4.2.4 Frequency and Phase Modulator

The *frequency modulation* (FM) is widely used for broadcasting and has found interesting applications for sound synthesis [Roa96]. The continuous-time description of an angle modulated carrier signal is given by

$$x_{PM/FM}(t) = A_c \cos[2\pi f_c t + \phi(t)] \quad (4.6)$$

where A_c is the amplitude of the signal and the argument of the cosine is given by carrier frequency f_c and the modulating signal $m(t)$ according to

$$\phi(t) = k_{PM} \cdot m(t) \quad (4.7)$$

$$= 2\pi \cdot k_{FM} \cdot \int_{-\infty}^t m(\tau) d\tau. \quad (4.8)$$

For phase modulation (PM) the phase $\phi(t)$ is directly proportional to the modulating signal $m(t)$, while for frequency modulation the phase $\phi(t)$ is the integral of the modulating signal $m(t)$. Some examples of frequency and phase modulation are shown in Fig. 4.7. In the first example the modulating signal is a sinusoid which shows that the resulting FM and PM signals are the same. The second example (c) and (d) shows the difference between FM and PM, where the modulating signal is now a bipolar pulse signal. The last example in (e) and (f) shows the result of a ramp type signal. The main idea behind using these techniques is the control of the carrier frequency by a modulating signal $m(n)$. We especially notice from Fig. 4.7 the possibility to change the carrier frequency by a sinusoid and a ramp signal.

Using angle modulation for audio effects is different from the previous discussion, where a modulating signal $m(n)$ is used to modify the phase $\phi(t)$ of a cosine of fixed carrier frequency f_c . By phase modulation we mean the direct modification of the phase of the audio signal by a control parameter or modulating signal $m(n)$. A phase modulator for an audio signal can be regarded as a system which performs a phase modulation of the audio input signal $x(n)$. The phase modulator system can be described by a time-variant impulse response $h(n)$ and leads to a phase modulated output signal $x_{PM}(n)$ according to

$$h(n) = \delta[n - m(n)] \quad (4.9)$$

$$y(n) = x_{PM}(n) = x(n) * h(n) = x(n) * \delta[n - m(n)]. \quad (4.10)$$

The result for phase modulation (PM) of the signal $x(n)$ can then be written as

$$y(n) = x_{PM}(n) = x(n - m(n)) \quad (4.11)$$

where $m(n)$ is a continuous variable, which changes every discrete time instant n . Therefore $m(n)$ is decomposed into an integer and a fractional part [Dat97]. The integer part is implemented by a series of M unit delays, the fractional part is approximated by interpolation filters [Dat97, LVKL96, Zöl97], e.g. linear interpolation, spline interpolation or allpass interpolation (see Fig. 4.8). The discrete-time

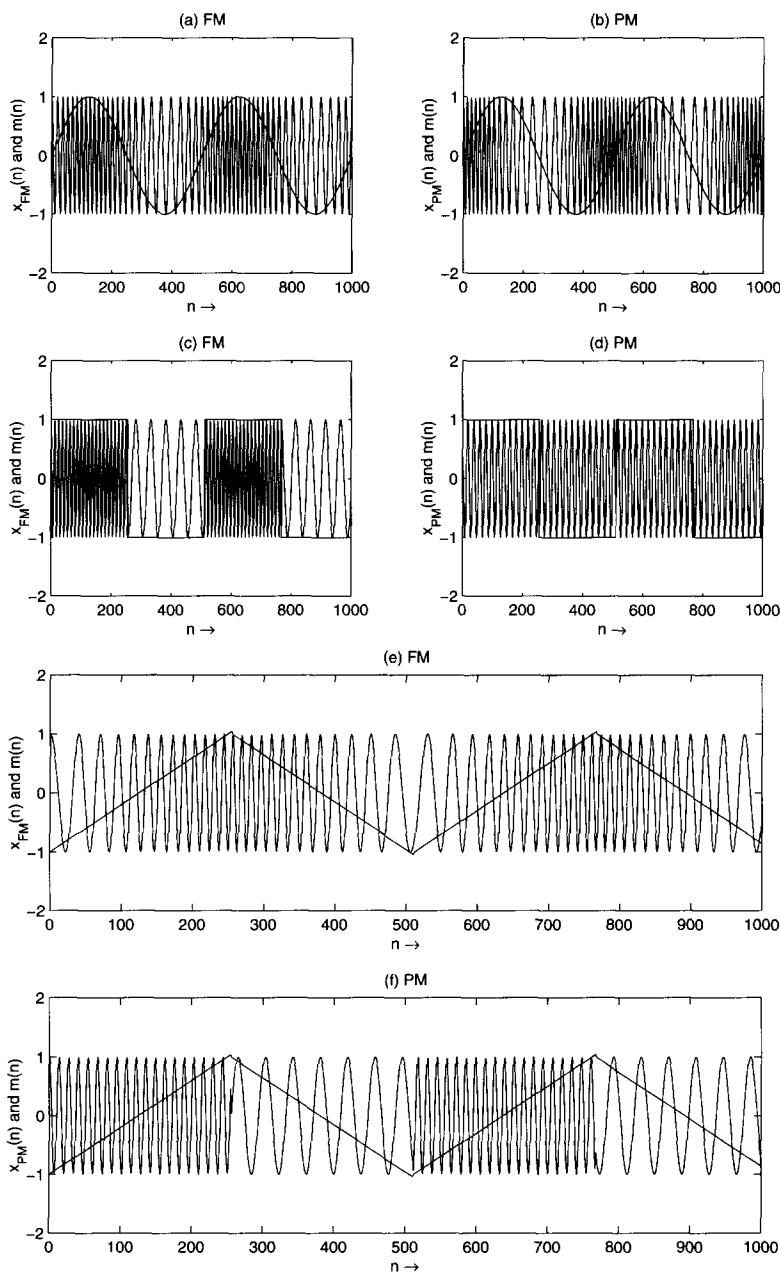


Figure 4.7 Examples of angle modulation.

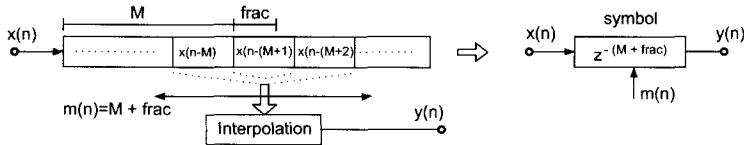


Figure 4.8 Phase modulation by delay line modulation.

Fourier transform of (4.11) yields

$$Y(e^{j\Omega}) = X_{PM}(e^{j\Omega}) = X(e^{j\Omega})e^{-j\Omega m(n)}, \quad (4.12)$$

which shows the phase modulation of the input signal by a time-variant delay line $m(n)$. The phase modulation is performed by the modulating signal $m(n)$.

For sine type modulation, useful for vibrato effects, the modulation signal can be written as

$$m(n) = M + \text{DEPTH} \cdot \sin(\omega_M n T). \quad (4.13)$$

For a sinusoidal input signal the so-called resampling factor for sine type modulation can be derived as

$$\alpha(n) = \frac{\omega_I}{\omega} = 1 - \text{DEPTH} \cdot \omega_M T \cos(\omega_M n T). \quad (4.14)$$

The instantaneous radian frequency is denoted by ω_I and the radian frequency of the input sinusoid is denoted by ω . The resampling factor is regarded as the pitch change ratio in [Dat97]. For sine type modulation the mean value of the resampling factor $\alpha(n)$ is one. The consequence is an output signal, which has the same length as the input signal, but has a vibrato centered around the original pitch.

For ramp type modulation according to

$$m(n) = M \pm \text{SLOPE} \cdot n, \quad (4.15)$$

the resampling factor $\alpha(n)$ for the sinusoidal input signal is given by

$$\alpha(n) = \frac{\omega_I}{\omega} = 1 \mp \text{SLOPE}. \quad (4.16)$$

The output signal is pitch transposed by a factor α and the length of the output data is altered by the factor $1/\alpha$. This behavior is useful for pitch transposing applications.

4.3 Demodulators

Each modulation has a suitable demodulation scheme. The demodulator for the ring modulator uses exactly the same scheme, so no new effect is to be expected there. The demodulator for the amplitude detector is called an amplitude follower in the realm of digital audio effects. Several schemes are available, some are inspired from analog designs, some are much easier to realize using digital techniques. These demodulators are comprised of three parts: a detector, an averager and a scaler.

4.3.1 Detectors

The detector can be a half-wave rectifier $d_h(t)$, a full-wave rectifier $d_f(t)$, a squarer $d_r(t)$ or an instantaneous envelope detector $d_i^2(t)$. The first two detectors are directly inspired by analog designs. They are still useful to achieve effects having typical analog behavior. The third and fourth types are much easier to realize in the digital domain (Figure 4.9). The four detectors are computed as follows:

$$\begin{aligned}
 x(n) & \quad \text{input signal} \\
 \hat{x}(n) & \quad \text{Hilbert transform of } x(n) \\
 d_h(n) & = \max[0, x(n)] \\
 d_f(n) & = |x(n)| \\
 d_r(n) & = x^2(n) \\
 d_i^2(n) & = x^2(n) + \hat{x}^2(n)
 \end{aligned} \tag{4.17}$$

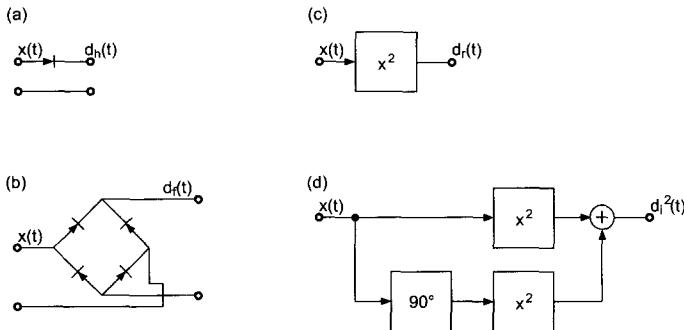


Figure 4.9 Detectors: (a) half-wave, (b) full-wave, (c) squarer, (d) instantaneous envelope.

4.3.2 Averagers

In the analog domain, the averager is realized with a resistor-capacitor (RC) network and in the digital domain using a first order lowpass filter. Both structures are characterized by a time-constant τ . The filter is implemented as:

$$\begin{aligned}
 g & = \exp[-1/(f_s\tau)] \\
 d(n) & = \text{detector output} \\
 y(n) & = (1-g)d(n) + gy(n-1)
 \end{aligned} \tag{4.18}$$

The time-constant must be chosen in accordance with the application. A short time-constant is suitable when fast variations of the input signal must be followed. A larger time-constant is better to measure the long-term amplitude of the input signal. This averager is nevertheless not suitable for many applications. It is often

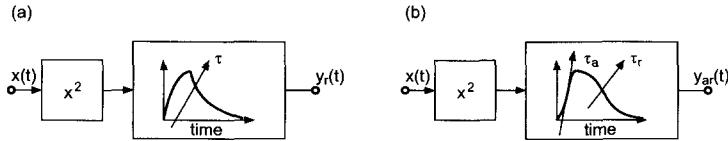


Figure 4.10 RMS (Root Mean Square)) detectors. (a) single time-constant; (b) attack and release time-constants.

necessary to follow short attacks of the input signal. This calls for a very small time-constant, 5 ms typically. The output of the averager will then react very fast to any amplitude variation, even to the intrinsic variations within a period of a low frequency signal. We understand that we need an averager with two time-constants: an attack time-constant τ_a and a release time-constant τ_r . To distinguish it from the basic averager, we will call this one the *AR-averager*. McNally has proposed an implementation having two fixed coefficients [McN84, Zöl97] and Jean-Marc Jot has an alternative where a single coefficient is varied according to the relationship between the input and the output of the averager (Figure 4.10):

$$\begin{aligned} g_a &= \exp[-1/(f_s \tau_a)] \\ g_r &= \exp[-1/(f_s \tau_r)] \\ d(n) &\text{ detector output} \end{aligned} \tag{4.19}$$

$$\left\{ \begin{array}{l} \text{if } y_{ar}(n-1) < d(n) \text{ then } g = g_a \text{ else } g = g_r \\ y_{ar}(n) = (1-g)d(n) + gy_{ar}(n-1) \end{array} \right. \tag{4.20}$$

4.3.3 Amplitude Scalers

The outputs of the systems described above are all different. In order to get measures that are comparable with each other, it would be necessary to scale the outputs. Although scaling schemes are typically defined for sine waves, each type of signal will require a different scaling scheme. To build a RMS detector or an instantaneous envelope detector, a root extractor would still be necessary, but building an accurate device can be difficult in analog and computationally intensive in digital. Fortunately, it is often possible to avoid the root extraction by modifying the circuit that makes use of the averager output, so that it works fine with squared measures. For these practical reasons the scaling is taken into account most of the time within the device that follows the averager output. If this device is a display, then the scaling can be done by changing the display marks.

4.3.4 Typical Applications

Well-known devices or typical applications relate to the previous schemes as follows:

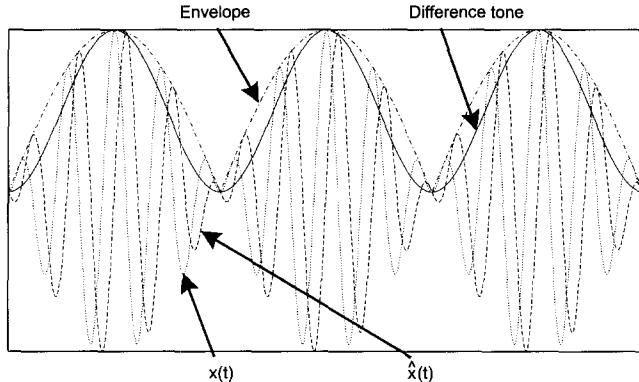


Figure 4.11 Instantaneous envelope detector as applied to detect a difference tone that is produced by two sine waves.

- The AM-detector comprises the half-wave rectifier and of the basic averager.
- The Volume-Meter (VU-meter) is an AM-detector. It measures the average amplitude of the audio signal.
- The Peak Program Meter (PPM) is, according to DIN45406, a full-wave rectifier followed by an AR-averager with 10 ms integration-time and 1500 ms release-time.
- The RMS detector, as found in electronic voltmeters, uses the squarer and the basic averager.
- A sound level meter uses a RMS detector along with an AR-averager to measure impulsive signals.
- The RMS detector associated with an AR-averager is the best choice for amplitude follower applications in vocoders, computer music and live-electronics [Dut98b, m-Fur93].
- Dynamics processors use various types of the above mentioned schemes in relation to the effect and to the quality that has to be achieved.
- The instantaneous envelope detector, without averager, is useful to follow the amplitude of a signal with the finest resolution. The output contains typically audio band signals. A particular application of the $d_i^2(t)$ detector is the amplification of difference tones (Figure 4.11) [Dut96, m-MBa95].

4.4 Applications

Several applications of modulation techniques for audio effects are presented in the literature [Dut98a, War98, Dis99]. We will now summarize some of these effects.

4.4.1 Vibrato

The cyclic variation of the pitch of the input signal is the basic application of the phase modulator described in the previous section (see Fig. 4.12). The variation is controlled by a low-frequency oscillator.

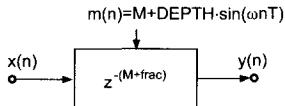


Figure 4.12 Vibrato based on a phase modulator.

4.4.2 Stereo Phaser

The application of a SSB modulator for a stereo phaser is described in [War98]. Figure 4.13 shows a SSB modulator performed by a recursive allpass implementation of a Hilbert filter. The phase difference of 90° is achieved through special designed allpass filters. A further effect with this approach is a rotating speaker effect, if you connect the output signals $y_L(n)$ and $y_R(n)$ via DACs to loudspeakers.

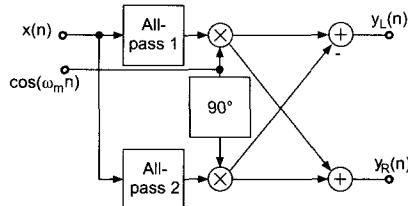


Figure 4.13 Stereo phaser based on SSB modulation [War98].

4.4.3 Rotary Loudspeaker Effect

Introduction

The rotary loudspeaker effect was first used for the electronic reproduction of organ instruments. Figure 4.14 shows the configuration of a rotating bi-directional loudspeaker horn in front of a listener. The sound in the listener's ears is altered by the Doppler effect, the directional characteristic of the speakers, and phase effects due to air turbulence. The Doppler effect raises and lowers the pitch according to the rotation speed. The directional characteristic of the opposite horn arrangement performs an intensity variation in the listener's ears. Both the pitch modification and the intensity variation are performed by speaker A and in the opposite direction by speaker B.

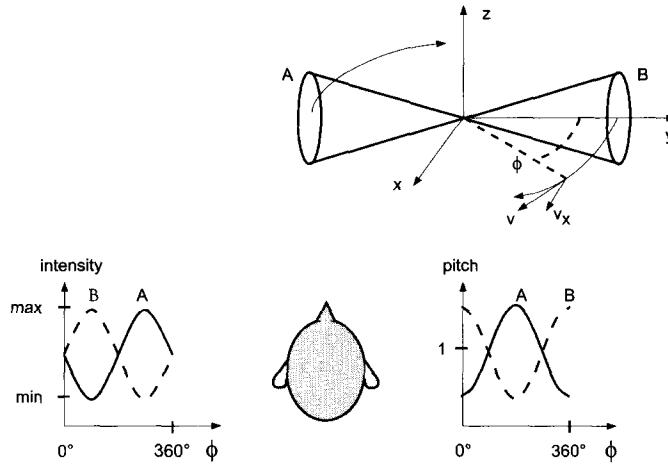


Figure 4.14 Rotary loudspeaker [DZ99].

Signal Processing

A combination of modulation and delay line modulation can be used for a rotary loudspeaker effect simulation [DZ99], as shown in Fig. 4.15. The simulation makes use of a modulated delay line for pitch modifications and amplitude modulation for intensity modifications. The simulation of the Doppler effect of two opposite horns is done by the use of two delay lines modulated with 180 phase shifted signals in vibrato configuration (see Fig. 4.15). A directional sound characteristic similar to rotating speakers can be achieved by amplitude modulating the output signal of the delay lines. The modulation is synchronous to the delay modulation in a manner, that the back moving horn has lower pitch and decreasing amplitude. At the return point the pitch is unaltered and the amplitude is minimum. The movement in direction to the listener causes a raised pitch and increasing amplitude. A stereo rotary speaker effect is perceived due to unequal mixing of the two delay lines to the left and right channel output.

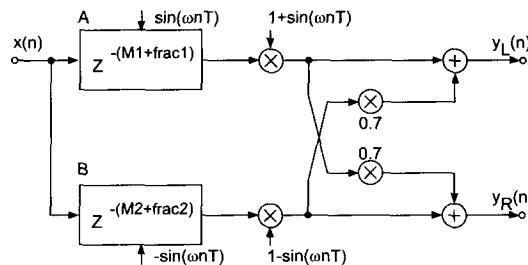


Figure 4.15 Rotary loudspeaker simulation [DZ99].

Musical Applications

By imprinting amplitude and pitch modulations as well as some spatialization, this effect makes the sounds more lively. At lower rotation speeds it is reminiscent of the echoes in a cathedral whereas at higher rotation speeds it gets a ring modulation flavor. This effect is known as “Leslie” from the name of Donald E. Leslie, who invented it in the early forties. It was licensed to electronic organ manufacturers such as Baldwin, Hammond or Wurlizer but it has also found applications for other musical instruments such as the guitar or even the voice (“Blue Jay Way” on the Beatles LP “Magical Mystery Tour” [Sch94].) A demonstration of a Leslie simulator can be heard on [m-Pie99]. This effect can also be interpreted as a rotating microphone between two loudspeakers. You may also imagine that you are sitting on a merry-go-round and you pass by two loudspeakers.

4.4.4 SSB Effects

Single-sideband modulation can be used as a *special effect* for detuning of percussion instruments or voices. The harmonic frequency relations are modified by using this technique. Another application is time-variant filtering: first use SSB modulation to shift the input spectrum, then apply filtering or phase modulation and then perform the demodulation of the signal, as shown in Fig. 4.16 [Dis99, DZ99]. The frequency shift of the input signal is achieved by a low-frequency sinusoid. Arbitrary filters can be used in between modulation and demodulation. The simulation of the mechanical vibrato bar of an electric guitar can be achieved by applying a vibrato instead of a filter [DZ99]. Such a vibrato bar alters the pitch of the lower strings of the guitar in larger amounts than the higher strings and thus a non-harmonic vibrato results. The SSB approach can also be used for the construction of modified flangers.

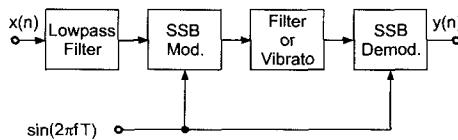


Figure 4.16 SSB modulation-filtering-demodulation: if a vibrato is performed instead of the filter a mechanical vibrato bar simulation is achieved.

Further applications of SSB modulation techniques for audio effects are presented in [Dut98a, War98].

4.4.5 Simple Morphing: Amplitude Following

Among the many different meanings that the word “morphing” can have, let us now consider its first meaning: imposing a feature of one sound onto another. The amplitude envelope, the spectrum as well as the time structure are features that can

be morphed. Morphing the amplitude envelope can be achieved by the amplitude follower whereas morphing a spectrum or a time structure can be achieved by the use of convolution.

Introduction

Envelope following is one of the various methods developed to breathe more life into synthetic sounds. The amplitude envelope of a control signal, usually coming from a real acoustical source, is measured and used to control the amplitude of the synthetic sounds. For example, the amplitude envelope of speech can be used to control the amplitude of broadband noise. Through this process the noise seems to have been articulated like voice. A refinement of this method has led to the development of the vocoder where the same process is applied in each of the frequency bands in which the voice as well as the noise are divided.

An audio effect is achieved when the amplitude of an input signal is modified by a predefined amplitude envelope or according to the amplitude of another signal. In the latter case the process is called amplitude following.

Signal Processing

If an amplitude envelope is used, the input signal is multiplied by the output of the envelope generator. If a control signal is used, its envelope has to be measured before it can be multiplied with the input signal. When an accurate measurement is desired, a RMS detector should be used. However, signals from acoustic instruments have usually fairly limited amplitude variations and their loudness variations are more dependent on spectrum modifications than on amplitude modifications. If the loudness of the output signal has to be similar to that of the controlling signal, then an expansion of the dynamic of the controlling signal should be performed. An effective way to expand the dynamic by a factor 2 is to eliminate the root extraction from the scaler and use a much simpler MS (Mean Square) detector.

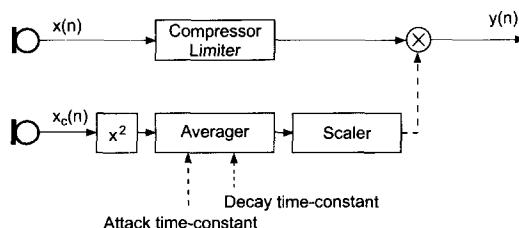


Figure 4.17 The amplitude of an input signal $x(n)$ is controlled by that of another signal $x_c(n)$. The amplitude of the input signal is first leveled before the modulation by the amplitude measured on the controlling signal.

Musical Applications and Control

In “Swim, swan”, Kiyoshi Furukawa has extended the sound of a clarinet by additional synthetic sounds. In order to link these sounds intimately to the clarinet, their amplitude is controlled by that of the clarinet. In this case, the input sound is the synthetic sound and the controlling sound is the clarinet. The mixing of the synthetic sounds with the clarinet is done in the acoustic domain of the performance space [m-Fur93].

The amplitude variations of the controlling signal applied to the input signal produce an effect that is perceived in the time domain or in the frequency domain, according to the frequency content of the modulating signal. For sub-audio rates (below 20 Hz) the effect will appear in the time domain and we will call it “amplitude following”, whereas for audio modulation rates (above 20 Hz), the effect will be perceived in the frequency domain and will be recognized as an amplitude modulation.

If the control signal has a large bandwidth, the spectrum of the amplitude will have to be reduced by the averager. Typical settings for the decay time constant of the averager are in the range of 30 to 100 ms. Such values will smooth out the amplitude signal so that it remains in the sub-audio range. However, it is often desired that the attacks, that are present in the control signal, are morphed onto the input signal as attacks and are not smoothed out by the averager. This is why it is recommended to use a shorter attack time constant than the decay time constant. Typical values are in the range of 1 to 30 ms.

The amplitude variations of the input signal could be opposite to those of the controlling signal, hence reducing the impact of the effect or be similar and provoke an expansion of the dynamic. In order to get amplitude variations at the output that are similar to those of the controlling signal, it is recommended to process the input signal through a compressor-limiter beforehand [Hal95, p. 40].

In his work “Diario polacco”, Luigi Nono specifies how the singer should move away from her microphone in order to produce amplitude modifications that are used to control the amplitude of other sounds [Hal95, p. 67-68].

Applying an amplitude envelope can produce interesting modifications of the input signal. Take, for example, the sustained sound of a flute and apply iteratively a triangular amplitude envelope. By varying the slopes of the envelope and the iteration rate, the original sound can be affected by a tremolo or a *Flatterzunge* and evoke percussive instruments [Dut88]. Such sound transformations are reminiscent of those (anamorphoses) that the early electroacoustic composers were fond of [m-Sch98].

4.5 Conclusion

In this chapter we introduced the concepts of modulators and demodulators in relation to digital audio effects. Although well known in communication engineering

and already successfully used for music synthesizers, the special emphasis on modulation and demodulation can also help to clarify the importance of these techniques in the field of audio effects. The interaction of modulators/demodulators with filters and delays is one of the fundamental processes for many audio effects occurring in the real world. Several applications of modulators and demodulators may serve as examples for experiments and further research.

Sound and Music

- [m-MBa95] M. Bach: 55 Sounds for Cello. Composition for cello and live-electronics. 1995.
- [m-Fur93] K. Furukawa: "Swim, swan", composition for clarinet and live-electronics. ZKM, 1993.
- [m-Hoe82] K. Hörmann and M. Kaiser: Effekte in der Rock- und Popmusik: Funktion, Klang, Einsatz. Bosse-Musik-Paperback ; 21, 1982. Sound examples. Cassette BE 2240 MC.
- [m-Pie99] F. Pieper: Leslie-Simulatoren. CD, Tr. 33. in *Das Effekte Praxisbuch*. Ch. 12. GC Carstensen, 1999.
- [m-Sch98] P. Schaeffer and G. Reibel: Solfège de l'objet sonore. Booklet + 3 CDs. First published 1967. INA-GRM, 1998.

Bibliography

- [Dat97] J. Dattoro. Effect design, part 2: Delay-line modulation and chorus. *J. Audio Eng. Soc.*, 45(10):764–788, October 1997.
- [De 00] G. De Poli. Personal communication, 2000.
- [Dis99] S. Disch. Digital audio effects — modulation and delay lines. Master's thesis, Technical University Hamburg-Harburg, 1999.
- [Dut88] P. Dutilleux. *Mise en œuvre de transformations sonores sur un système temps-réel*. Technical report, Rapport de stage de DEA, CNRS-LMA, June 1988.
- [Dut91] P. Dutilleux. *Vers la machine à sculpter le son, modification en temps réel des caractéristiques fréquentielles et temporelles des sons*. PhD thesis, University of Aix-Marseille II, 1991.
- [Dut96] P. Dutilleux. Verstärkung der Differenztöne (f2-f1). In *Bericht der 19. Tonmeistertagung Karlsruhe, Verlag K.G. Saur*, pp. 798–806, 1996.

- [Dut98a] P. Dutilleux. *Filters, Delays, Modulations and Demodulations: A Tutorial*. In *Proc. DAFX-98 Digital Audio Effects Workshop*, pp. 4–11, Barcelona, November 1998.
- [Dut98b] P. Dutilleux. Opéras multimédias, le rôle des ordinateurs dans trois créations du zkm. in musique et arts plastiques. In *GRAME et Musée d'Art contemporain, Lyon*, pp. 73–79, 1998.
- [DZ99] S. Disch and U. Zölzer. Modulation and delay line based digital audio effects. In *Proc. DAFX-99 Digital Audio Effects Workshop*, pp. 5–8, Trondheim, December 1999.
- [Hal95] H.P. Haller. *Das Experimental Studio der Heinrich-Strobel-Stiftung des Südwestfunks Freiburg 1971–1989, Die Erforschung der Elektronischen Klangumformung und ihre Geschichte*. Nomos, 1995.
- [LVKL96] T.I. Laakso, V. Välimälki, M. Karjalainen, and U.K. Laine. Splitting the unit delay. *IEEE Signal Processing Magazine*, 13:30–60, 1996.
- [McN84] G.W. McNally. Dynamic range control of digital audio signals. *J. Audio Eng. Soc.*, 32(5):316–327, May 1984.
- [Orf96] S.J. Orfanidis. *Introduction to Signal Processing*. Prentice-Hall, 1996.
- [Roa96] C. Roads. *The Computer Music Tutorial*. MIT Press, 1996.
- [Sch94] W. Schiffner. *Rock und Pop und ihre Sounds*. Elektor-Verlag, 1994.
- [Ste87] M. Stein. *Les modems pour transmission de données*. Masson CNET-ENST, 1987.
- [Vid91] A. Vidolin. Musical interpretation and signal processing. In G. De Poli, A. Piccialli, and C. Roads (eds), *Representations of Musical Signals*, pp. 439–459. MIT Press, 1991.
- [War98] S. Wardle. A Hilbert-transformer frequency shifter for audio. In *Proc. DAFX-98 Digital Audio Effects Workshop*, pp. 25–29, Barcelona, November 1998.
- [Zöl97] U. Zölzer. *Digital Audio Signal Processing*. John Wiley & Sons, Ltd, 1997.

Chapter 5

Nonlinear Processing

P. Dutilleux, U. Zölzer

5.1 Introduction

Audio effect algorithms for dynamics processing, valve simulation, overdrive and distortion for guitar and recording applications, psychoacoustic enhancers and excitors fall into the category of nonlinear processing. They create intentional or unintentional harmonic or inharmonic frequency components which are not present in the input signal. Harmonic distortion is caused by nonlinearities within the effect device. Most of these signal processing devices are controlled by varying parameters and simultaneously listening to the output signal and monitoring the output signal by a level meter. A lot of listening and recording experience is necessary to obtain sound results which are preferred by most listeners. The application of these signal processing devices is an art of its own and of course one of the main tools for recording engineers and musicians.

Nonlinear processing/processors is the term for signal processing algorithms or signal processing devices in the analog or digital domains which deliver an output signal as a sum of sinusoids $y(n) = A_0 + A_1 \sin(2\pi f_1 Tn) + A_2 \sin(2 \cdot 2\pi f_1 Tn) + \dots + A_N \sin(N \cdot 2\pi f_1 Tn)$, if the input signal is a sinusoid of known amplitude and frequency according to $x(n) = A \sin(2\pi f_1 Tn)$. A linear system will deliver the output signal $y(n) = A_{\text{out}} \sin(2\pi f_1 Tn + \varphi_{\text{out}})$ which again is a sinusoid where the amplitude is modified by the magnitude response $|H(f_1)|$ of the transfer function according to $A_{\text{out}} = |H(f_1)| \cdot A$ and the phase response $\varphi_{\text{out}} = \varphi_{\text{in}} + \angle H(f_1)$ is modified by the phase $\angle H(f_1)$ of the transfer function. Block diagrams in Fig. 5.1 showing both input and output signals of a linear and a nonlinear system illustrate the difference between both systems. A measurement of the total harmonic distortion gives an

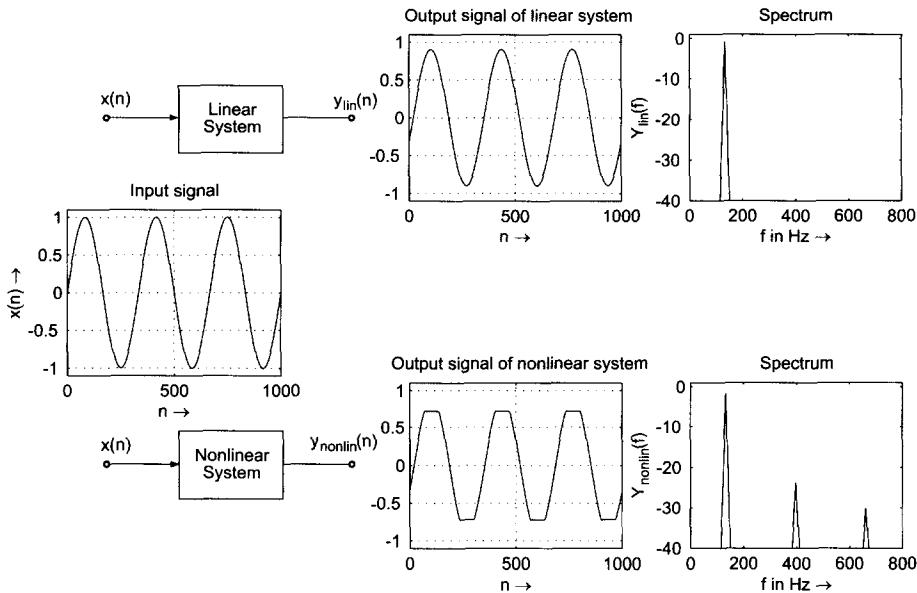


Figure 5.1 Input and output signals of a linear and nonlinear system. The output signal of the linear system is changed in amplitude and phase. The output signal of the nonlinear system is strongly shaped by the nonlinearity and consists of a sum of harmonics, as shown by the spectrum.

indication of the nonlinearity of the system. Total harmonic distortion is defined by

$$\text{THD} = \sqrt{\frac{A_2^2 + A_3^2 + \dots + A_N^2}{A_1^2 + A_2^2 + \dots + A_N^2}}, \quad (5.1)$$

which is the square root of the ratio of the sum of powers of all harmonic frequencies above the fundamental frequency to the power of all harmonic frequencies including the fundamental frequency.

We will discuss nonlinear processing in three main musical categories. The first category consists of dynamic range controllers where the main purpose is the control of the signal envelope according to some control parameters. The amount of harmonic distortion introduced by these control algorithms should be kept as low as possible. Dynamics processing algorithms will be introduced in section 5.2. The second class of nonlinear processing is designed for the creation of strong harmonic distortion such as guitar amplifiers, guitar effect processors, etc. and will be introduced in section 5.3. These nonlinear processors can be described by a linear part and a nonlinear part. The linear part consists of the impulse response and the nonlinear part is a combination of a nonlinear function $f[x(n)]$ of the input signal $x(n)$ and linear systems in front and behind this nonlinearity. The theory and simulation of nonlinear systems will be discussed in section 5.3.1. The third category can be described by the same theoretical approach and is represented by signal processing

devices called exciters and enhancers. Their main field of application is the creation of additional harmonics for a subtle improvement of the main sound characteristic. The amount of harmonic distortion is usually kept small to avoid a pronounced effect. Exciters and enhancers are based on psycho-acoustic fundamentals and will be discussed in section 5.4.

5.2 Dynamics Processing

Dynamics processing is performed by amplifying devices where the gain is automatically controlled by the level of the input signal. We will discuss limiters, compressors, expanders and noise gates. A good introduction to the parameters of dynamics processing can be found in [Ear76] (pp. 242-248).

Dynamics processing is based on an amplitude/level detection scheme sometimes called an envelope follower, an algorithm to derive a gain factor from the result of the envelope follower and a multiplier to weight the input signal (see Fig. 5.2). The envelope follower calculates the mean of the absolute values $|x(n)|$ over a predefined time interval. The output to input relation is usually described by the characteristic curve $y(n) = f[x(n)]$ as shown in Fig. 5.2. Certain thresholds are defined for a change of the output to input behavior. For the given example the output is limited to $y(n) = \pm 0.5$ for $|x(n)| > 0.5$ and $y(n) = x(n)$ for $|x(n)| \leq 0.5$. The lower path consisting of the envelope detector and the following processing to derive the gain factor $g(n)$ is usually call the *side chain* path. Normally, the gain factor is derived from the input signal, but the side chain path can also be connected to another signal for controlling the gain factor of the input signal.

Signal Processing

A detailed description of a dynamic range controller is shown in Fig. 5.3. It consists of a direct path for delaying the input signal and a side chain path. The side chain path performs a level measurement and a subsequent gain factor calculation which is then used as a gain factor for the delayed input signal. The level measurement is followed by a static function and a part for the attack and release time adjustment. Besides the time signals $x(n)$, $f(n)$, $g(n)$ and $y(n)$ the corresponding signal levels X , G and Y are denoted. These level values are the logarithm of the root mean square $x_{\text{rms}}(n)$ (RMS value) or peak value $x_{\text{peak}}(n)$ of the time signals according to $X = 20 \cdot \log_{10}(x)$. The multiplication $y(n) = g(n) \cdot x(n - D)$ at the output of the dynamic range controller can be regarded as an addition in the logarithmic domain. This means $Y = X + G$ in dB. The calculation of the time-variant gain factor $g(n)$ is usually performed with a logarithmic level representation, because the human sensitivity of loudness follows a logarithmic relation. The delay of D samples in the direct path allows for the time delay of the side chain processing, which is mainly made up of the level measurement and the attack and release time adjustments.

The static function for the output level versus the input level (in dB) and the calculations are shown in the left part of Fig. 5.4. Inside this representation the

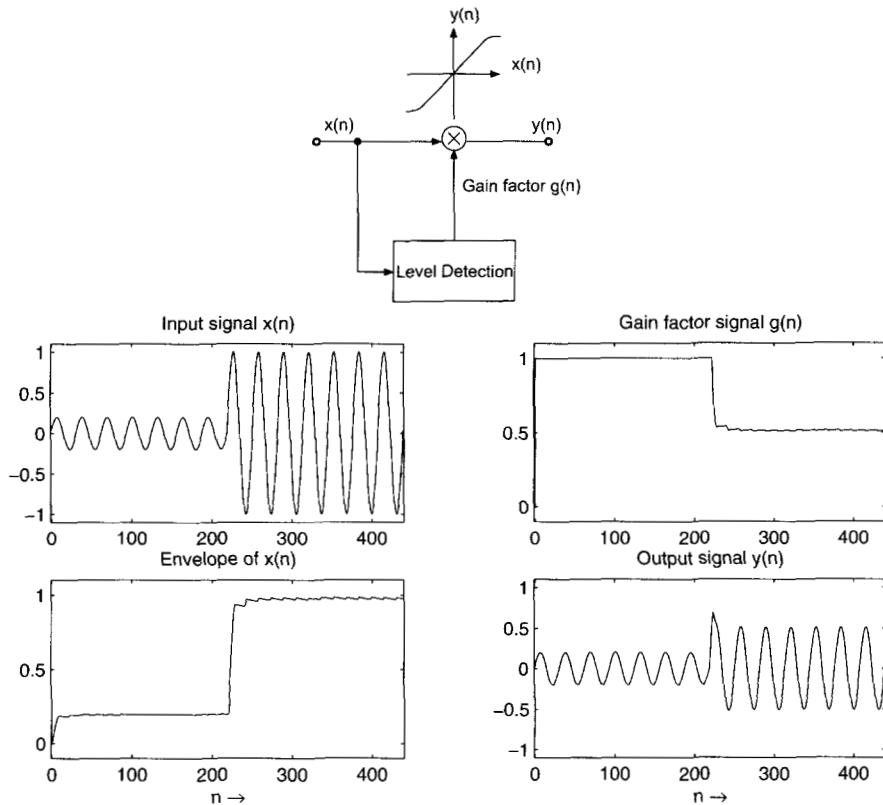


Figure 5.2 Block diagram of a nonlinear signal processing device with envelope detector. The lower plots show the input signal $x(n)$, the envelope of $x(n)$, the derived gain factor $g(n)$ and the output signal $y(n)$.

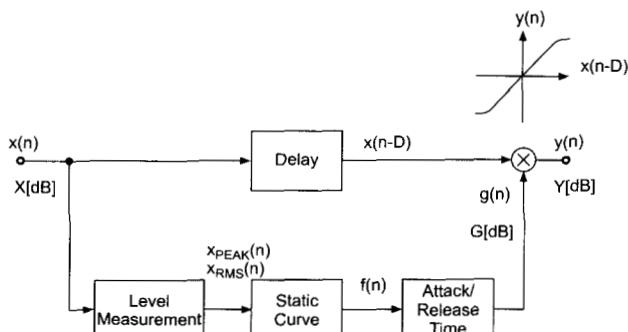


Figure 5.3 Block diagram of a dynamic range controller [Zöl97].

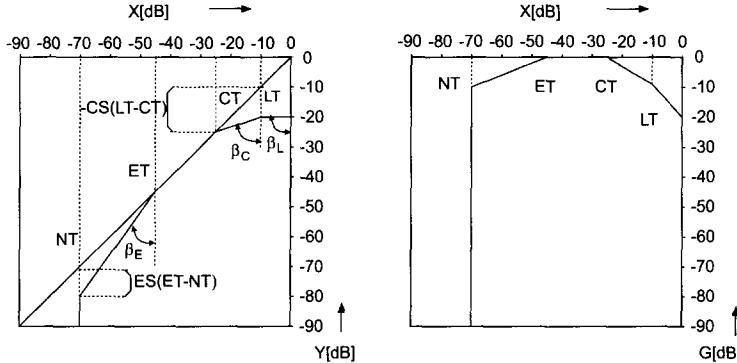


Figure 5.4 Static characteristic of a dynamic range controller [Zöl97].

thresholds for limiting (LT limiter threshold), compression (CT compressor threshold), expansion (ET expander threshold) and noise gate (NT noise gate threshold) are denoted. In the right part of Fig. 5.4 the input level versus the gain level (in dB) is shown which clearly shows the four regions of operation. For the description of the static function two further parameters, namely the slope factor $S = 1 - \frac{1}{R}$ and the compression factor $R = \frac{\Delta L_I}{\Delta L_O} = \frac{1}{1-S}$ are used. The compression factor R represents the fraction of input level change ΔL_I to output level change ΔL_O . With the help of Fig. 5.5 for compression the equation $Y = CT + \frac{1}{R}(X - CT)$ and the equation $R = \frac{Y-CT}{Y-CT} = \tan \beta_C$ can be derived.

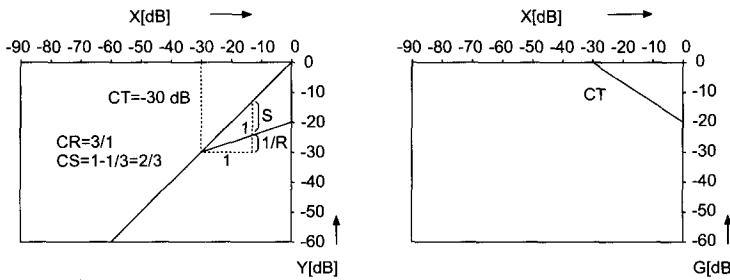


Figure 5.5 Static characteristic: definition of slope S and ratio R [Zöl97].

Typical values for compression factor and slope factor for the four regions of operation are:

Limiter	$R = \infty$	$S = 1$
Compressor	$1 < R < \infty$	$0 < S < 1$
Linear part	$R = 1$	$S = 0$
Expander	$0 < R < 1$	$-\infty < S < 0$
Noise gate	$R = 0$	$S = -\infty$

The calculation of the control parameter $f(n)$ in the logarithmic domain F in dB can be performed by simple line equations [Zöl97, RZ95] given by

$$\begin{aligned}\text{Limiter} \quad F_L &= -LS(X - LT) + CS(CT - LT) \\ \text{Compressor} \quad F_C &= -CS(X - CT) \\ \text{Linear part} \quad F_{\text{lin}} &= 0 \\ \text{Expander} \quad F_E &= -ES(X - ET) \\ \text{Noise gate} \quad F_{\text{NG}} &= -NS(X - NT) + ES(ET - NT).\end{aligned}$$

The dynamic behavior of a dynamic range controller is influenced by the level measurement approach (with attack AT and release time RT for peak measurement and averaging time TAV for RMS measurement) and further adjusted with special attack/release times which can be achieved by the systems shown in Figures 5.6 and 5.7.

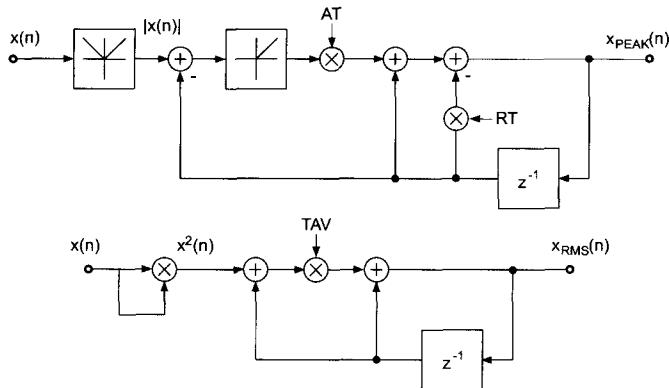


Figure 5.6 RMS and peak measurement (envelope detector/follower) for a dynamic range controller [McN84, Zöl97].

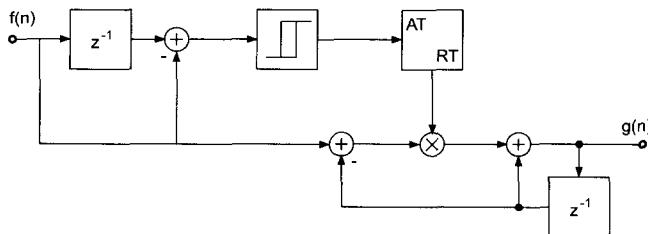


Figure 5.7 Dynamic filter: attack and release time adjustment for a dynamic range controller [McN84, Zöl97].

These envelope detectors/followers can also be used for other musical applications. The calculation of the attack time parameter is carried out by

$$AT = 1 - e^{-2.2T/t_{\text{AT}}},$$

where t is the time parameter in seconds and T is the sampling period. The release time parameter RT and the averaging parameter TAV can be computed by the same formula by replacing t_{AT} by t_{RT} or t_{TAV} , respectively. Further details and derivations can be found in [McN84, Zöl97]. The output factor $f(n)$ of the static function is used as the input signal to the dynamic filter with attack and release times in Fig. 5.7. The output signal $g(n)$ is the gain factor for weighting the delayed input signal $x(n - D)$ as shown in Fig. 5.3. In the following sections some special dynamic range controllers are discussed in detail.

5.2.1 Limiter

The functional units of a limiter are shown in Fig. 5.8. The purpose of the limiter is to provide control over the high peaks in the signal and to change the dynamics of the signal as little as possible. A limiter makes use of peak level measurement and should react very quickly to extensions of the limiter threshold. Typical parameters for a limiter are $t_{AT} = 0.02 \dots 0.04 \dots 10.24$ msec and $t_{RT} = 1 \dots 130 \dots 5000$ msec for the peak measurement and $t_{AT} = 0.02 \dots 0.04 \dots 10.24$ msec and $t_{RT} = 1 \dots 130 \dots 5000$ msec for the attack/release time adjustment. The fast attack and release of a limiter allow the volume reduction as soon as the signal crosses the threshold. By lowering the peaks, the overall signal can be boosted. Beside limiting single instrument signals, limiting is also often performed on the final mix of a multichannel application.

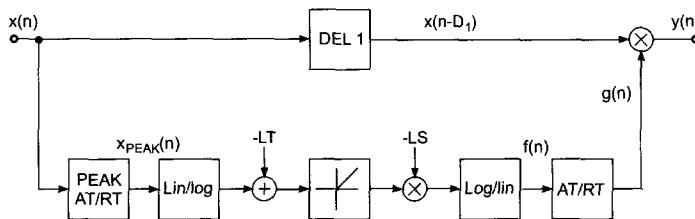


Figure 5.8 Block diagram of a limiter [Zöl97].

The following M-file 5.1 may serve as an example of a sample-based limiter implementation.

```
M-file 5.1 (limiter.m)
% Limiter.m
anzahl=220;
for n=1:anzahl,
    x(n)=0.2*sin(n/5);
end;
for n=anzahl+1:2*anzahl;
    x(n)=sin(n/5);
end;
```

```

slope=1;
tresh=0.5;
rt=0.01;
at=0.4;

xd(1)=0;
for n=2:2*anzahl;
    a=abs(x(n))-xd(n-1);
    if a<0, a=0; end;
    xd(n)=xd(n-1)*(1-rt)+at*a;
    if xd(n)>tresh,
        f(n)=10^(-slope*(log10(xd(n))-log10(tresh)));
        % linear calculation of f=10^(-LS*(X-LT))
    else f(n)=1;
    end;
    y(n)=x(n)*f(n);
end;

```

Figure 5.9 demonstrates the behavior of the time signals inside a limiter configuration of Fig. 5.8.

A variant for a hard limiter is given by the following M-file 5.2 which makes use of lowpass filtering the absolute value $|x(n)|$ by a second order Butterworth filter to obtain the gain factor $g(n)$ [Ben97].

```

M-file 5.2 (hard_limiter.m)
function y=hard_limiter(x, limit)
% Hard sound limiter, limit - normalized limit
g=filter(1e-5*[0.45535887 0.91071773849 0.45535887],...
...[1 -1.99395528 0.993973494],abs(x));
% detects the envelope of the signal with a second order
% Butterworth filter, cut off frequency 30 Hz
h=h/max(h);
for n=1:length(x)
    if h(n)>limit %if the signal envelope is above the limit
        x(n)=x(n)*limit/g(n);
    end;
end;
y=x;

```

5.2.2 Compressor and Expander

A dynamic range controller for compression and expansion is shown in Fig. 5.10. The gain factor calculation is based on an RMS measurement and some computations in the logarithmic domain. Typical parameters for compressors and expanders are $t_{AT} = 5$ msec and $t_{RT} = 130$ msec for the RMS measurement and $t_{AT} =$

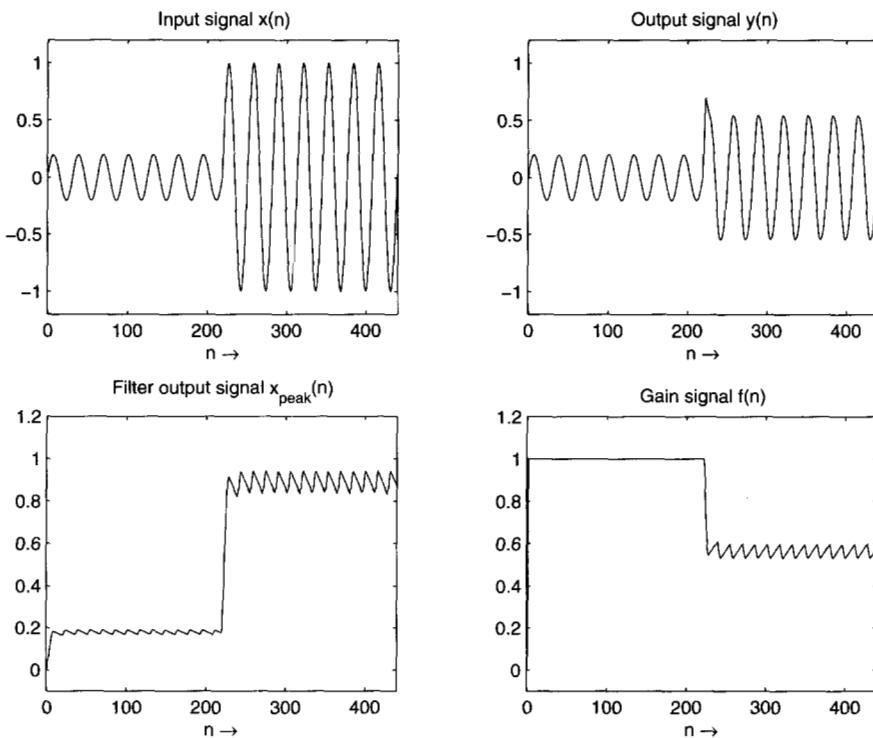


Figure 5.9 Signals for limiter.

$0.16 \dots 5 \dots 2600$ msec and $t_{RT} = 1 \dots 130 \dots 5000$ msec for the attack/release time adjustment. The programming is similar to the implementation for the limiter from the previous subsection.

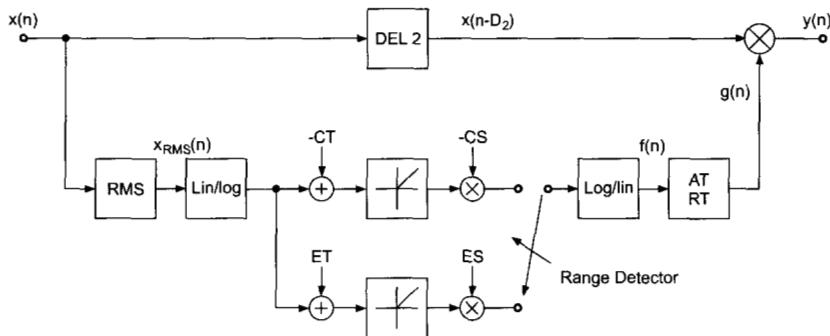


Figure 5.10 Block diagram of a compressor/expander [Zöl97].

A different method of implementation is given by [Ben97], which follows the

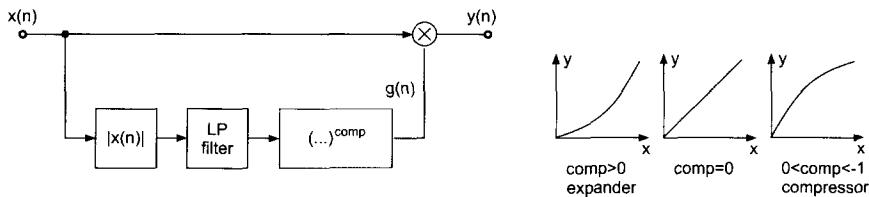


Figure 5.11 Block diagram of a compressor/expander [Ben97].

block diagram in Fig. 5.11. The corresponding M-file 5.3 illustrates the implementation.

```
M-file 5.3 (compexp.m)
function y=compexp(x,comp,release,attack,a,Fs)
% Compressor/expander
% comp - compression: 0>comp>-1, expansion: 0<comp<1
% a      - filter parameter <1
h=filter([(1-a)^2],[1.0000 -2*a a^2],abs(x));
h=h/max(h);
h=h.^comp;
y=x.*h;
y=y.*max(abs(x))/max(abs(y));
```

Compressors are used for reducing the dynamics of the input signal. Quiet parts or low levels of a signal are not modified but high levels or loud parts are reduced according to the static curve. The result is that the difference between the loud and quiet parts is lessened, and thus the overall signal level can be boosted, and thus the signal is made louder. Expanders operate on low level signals and increase the dynamics of these low level signals. This leads to a lively sound characteristic.

5.2.3 Noise Gate

The functional units of a noise gate are shown in Fig. 5.10. The decision to activate the gate is based on a peak measurement which leads to a fade in/fade out of the gain factor $g(n)$ with appropriate attack and release times. The input to the time constant system is set to zero if the input level falls below the noise gate threshold, and is set to one if the input level exceeds the noise gate level. The M-file 5.4 demonstrates an implementation with a hold time [Ben97].

```
M-file 5.4 (noisegt.m)
function y=noisegt(x,holdtime,ltrhold,utrhold,release,attack,a,Fs)
%y=noisegt(x,holdtime,ltrhold,utrhold,release,attack,a,Fs);
% noise gate with hysteresis
% holdtime - time in seconds the sound level has to be below the
%           threshhold value before the gate is activated
```

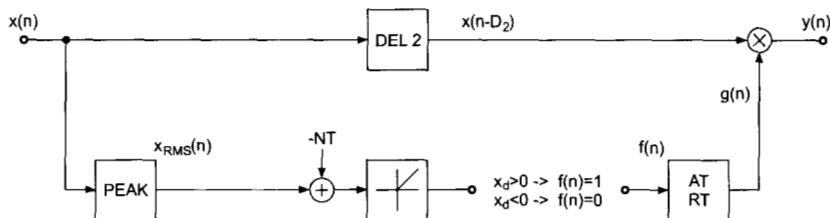


Figure 5.12 Block diagram of a noise gate [Zöl97].

```
% ltrhold - threshold value for activating the gate
% utrhold - threshold value for deactivating the gate > ltrhold
% release - time in seconds before the sound level reaches zero
% attack - time in seconds before the output sound level is the
%         same as the input level after deactivating the gate
% a      - pole placement of the envelope detecting filter <1
% Fs    - sampling frequency
rel=round(release*Fs); %number of samples for fade
att=round(attack*Fs); %number of samples for fade
g=zeros(size(x));
lthcnt=0;
uthcnt=0;
ht=round(holdtime*Fs);
h=filter([(1-a)^2],[1.0000 -2*a a^2],abs(x));%envelope detection
h=h/max(h);
for i=1:length(h)
    if (h(i)<=ltrhold) | ((h(i)<utrhold) & (lthcnt>0))
        % Value below the lower threshold?
        lthcnt=lthcnt+1;
        uthcnt=0;
        if lthcnt>ht
            % Time below the lower threshold longer than the hold time?
            if lthcnt>(rel+ht)
                g(i)=0;
            else
                g(i)=1-(lthcnt-ht)/rel; % fades the signal to zero
            end;
        elseif ((i<ht) & (lthcnt==i))
            g(i)=0;
        else
            g(i)=1;
        end;
    elseif (h(i)>=utrhold) | ((h(i)>ltrhold) & (uthcnt>0))
        % Value above the upper threshold or is the signal being faded in?
        uthcnt=uthcnt+1;
    end;
end;
```

```

if (g(i-1)<1)
% Has the gate been activated or isn't the signal faded in yet?
    g(i)=max(uthcnt/att,g(i-1));
else
    g(i)=1;
end;
lthcnt=0;
else
    g(i)=g(i-1);
    lthcnt=0;
    uthcnt=0;
end;
end;
y=x.*g;
y=y*max(abs(x))/max(abs(y));

```

The main use of a noise gate is to eliminate noise when the desired signal is not present. The noise gate attenuates only the soft signals. A particular application is found when recording a drum set. Each element of the drum set has a different decay time. When they are not manually damped, their sounds mix together and the result is no longer distinguishable. When each element is processed by a noise gate, every sound can automatically be faded out after the attack part of the sound. This results in an overall cleaner sound.

Further implementations of limiters, compressors, expanders and noise gates can be found in [Orf96] and in [Zö197] where special combined DRCs are also discussed.

5.2.4 De-esser

A de-esser is a signal processing device for processing speech and vocals. It consists of a bandpass filter tuned to the frequency range between 2-6 kHz to detect the level of the signal in this frequency band. If a certain threshold is exceeded, the amount of gain is used to control the gain factor of a peak- or notch-filter tuned to the same frequency band. The peak- or notch-filter is in the direct signal path (see Fig. 5.13). As an alternative to the bandpass/notch filters, highpass and shelving filters are used with good results. In order to make the de-esser more robust against input level changes the threshold should depend on the overall level of the signal - that is, a relative threshold [Nie00].

Applications of de-essers are mainly in the field of speech and vocal processing to avoid high frequency sibilance. Here, quite fast time constants are used. Another application is the feedback reduction in sound reinforcement systems where adaptively changing notch frequencies and gain factors are required.

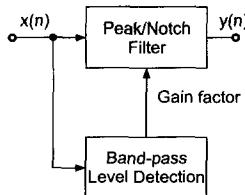


Figure 5.13 Block diagram of a de-esser.

5.2.5 Infinite Limiters

In order to catch overshoots from a compressor and limiter, a clipper - or infinite limiter - may be used [Nie00]. Another reason for using an infinite limiter is that the average level rises. The infinite limiter is a nonlinear operation working directly on the waveform by flattening the signal above a threshold. The simplest one is hard clipping which generates lots of high order harmonics. A gentler infinite limiter is the soft clipper which rounds the signal shape before the absolute clipping threshold. The rounding typically consists of a low order polynomial and therefore the harmonic spectrum rolls off faster [Nie00].

Due to the fact that the signal processing takes place in the digital domain, not only harmonic distortion but also aliasing distortion is generated. Aliasing distortion sounds metallic and is not really good. Although infinite limiting should be avoided during mix down of multi-channel recordings and recording sessions, several CDs make use of infinite limiting or saturation (see wave file in Fig. 5.14 and listen to Carlos Santana/Smooth [m-San99]).

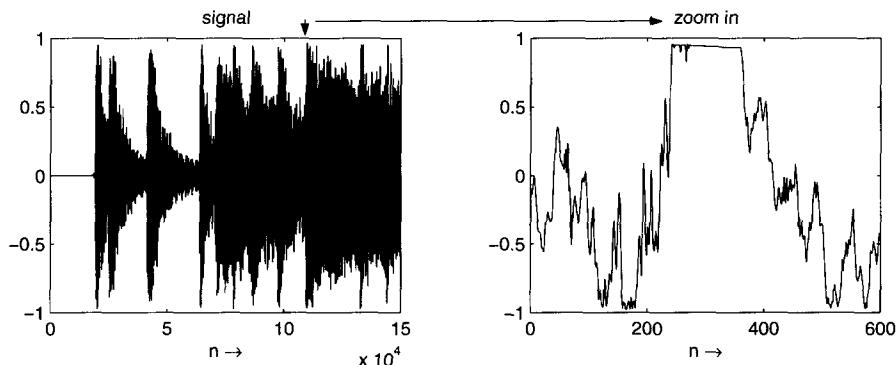


Figure 5.14 Infinite limiting (Santana - “Smooth”).

5.3 Nonlinear Processors

There are two approaches towards nonlinear processing for audio applications. The first approach is driven by musical applications of nonlinear effects and the second is driven by the reduction of nonlinear behavior especially in the field of loudspeakers. We cover the first approach of nonlinear effects for musical processing, where topics from nonlinear modeling of physical systems play an important role. Therefore, we investigate approximation and simulation techniques for nonlinear processors and will show simple implementation schemes for nonlinear processors. Special nonlinear waveshaping techniques for sound synthesis can be found in [Bru79, Arf79, De 84].

5.3.1 Basics of Nonlinear Modeling

Digital signal processing is mainly based on linear time-invariant systems. The assumption of linearity and time invariance is certainly valid for a variety of technical systems, especially for systems where input and output signals are bounded to a specified amplitude range. In fact several analog audio processing devices have nonlinearities like valve amplifiers, analog effect devices, analog tape recorders, loudspeakers and at the end of the chain the human hearing mechanism. A compensation and the simulation of these nonlinearities need nonlinear signal processing and of course a theory of nonlinear systems. From several models for different nonlinear systems discussed in the literature the Volterra series expansion is a suitable approach, because it is an extension of the linear systems theory. Not all technical and physical systems can be described by the Volterra series expansion, especially systems with extreme nonlinearities. If the inner structure of a nonlinear system is unknown, a typical measurement set-up, as shown in Fig. 5.15, with a pseudo-random signal as the input signal and recording the output signal is used. Input and output signals allow the calculation of the linear impulse response $h_1(n)$ by cross-correlation and kernels (impulse responses) of higher order $h_2(n_1, n_2), h_3(n_1, n_2, n_3), \dots, h_N(n_1, \dots, n_N)$ by higher order cross-correlations. The linear impulse response $h_1(n)$ is a one-dimensional, $h_2(n_1, n_2)$ is a two-dimensional and $h_N(n_1, \dots, n_N)$ is an N -dimensional kernel. An exhaustive treatment of these techniques can be found in [Sch80]. These N kernels can be used

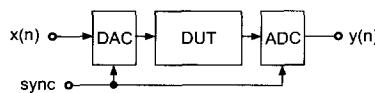


Figure 5.15 Measurement of nonlinear systems.

for an N -order Volterra system model given by

$$y(n) = \sum_{i=1}^N y_i(n) \quad (5.2)$$

$$\begin{aligned} &= \sum_{\nu_1=0}^{\infty} h_1(\nu_1)x(n - \nu_1)n + \\ &\quad \sum_{\nu_1=0}^{\infty} \sum_{\nu_2=0}^{\infty} h_2(\nu_1, \nu_2)x(n - \nu_1)x(n - \nu_2) + \\ &\quad \sum_{\nu_1=0}^{\infty} \sum_{\nu_2=0}^{\infty} \sum_{\nu_3=0}^{\infty} h_3(\nu_1, \nu_2, \nu_3)x(n - \nu_1)x(n - \nu_2)x(n - \nu_3) + \dots + \\ &\quad \sum_{\nu_1=0}^{\infty} \dots \sum_{\nu_N=0}^{\infty} h_N(\nu_1, \dots, \nu_N)x(n - \nu_1)\dots x(n - \nu_N). \end{aligned} \quad (5.3)$$

Figure 5.16 shows the block diagram representing (5.3).

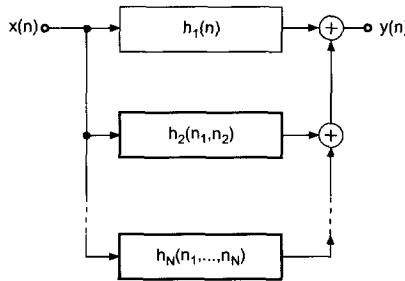


Figure 5.16 Simulation of nonlinear systems by an N -order Volterra system model.

A further simplification [Fra97] is possible if the kernels can be factored according to

$$h_i(n_1, n_2, \dots, n_i) = h_i^f(n_1)h_i^f(n_2)\dots h_i^f(n_i). \quad (5.4)$$

Then (5.3) can be written as

$$\begin{aligned} y(n) &= \sum_{\nu=0}^{\infty} h_1^f(\nu)x(n - \nu) + \left(\sum_{\nu=0}^{\infty} h_2^f(\nu)x(n - \nu) \right)^2 + \dots + \\ &\quad \left(\sum_{\nu=0}^{\infty} h_N^f(\nu)x(n - \nu) \right)^N, \end{aligned} \quad (5.5)$$

which is shown in block diagram representation in Fig. 5.17. This representation shows several advantages especially from the implementation point of view, because every subsystem can be realized by a one-dimensional impulse response or

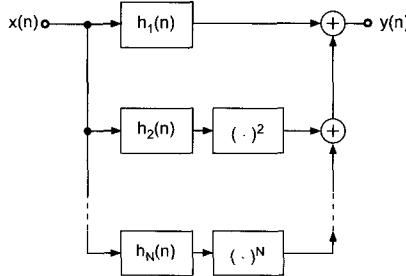


Figure 5.17 Simulation of nonlinear systems by an N -order Volterra system model with factored kernels.

the equivalent representations we have discussed in the previous chapters. At the output of each subsystem we have to perform the $(\cdot)^i$ operation on the corresponding output signals. The discussion so far can be applied to nonlinear systems with memory, which means that besides nonlinearities linear filtering operations are also included. Further material on nonlinear audio systems can be found in [Kai87, RH96, Kli98, FUB⁺98].

A simulation of a nonlinear system without memory, namely static nonlinear curves, can be done by a Taylor series expansion given by

$$y(n) = f[x(n)] = \sum_{i=0}^N b_i x^i(n). \quad (5.6)$$

Static nonlinear curves can be applied directly to the input signal, where each input amplitude is mapped to an output amplitude according to the nonlinear function $y = f[x(n)]$ (see Fig. 5.18). If one applies a squarer operation to the input signal of a given bandwidth, the output signal $y(n) = x^2(n)$ will double its bandwidth. As soon as the highest frequency after passing a nonlinear operation exceeds half the sampling frequency $f_s/2$, aliasing will fold this frequency back to the base band. In some effect applications additional aliasing distortions might be helpful, especially for extreme distortions in metal music. This means that for digital signals we first have to perform over-sampling of the input signal before applying any nonlinear operation to the input signal in order to avoid any aliasing distortions. This over-sampling is shown in Fig. 5.18 where first up-sampling is performed and then an interpolation filter H_I is used to suppress images up to the new sampling frequency Lf_s . Then the nonlinear operation can be applied followed by a band-limiting filter to $f_s/2$ and down-sampling.

As can be noticed, the output spectrum only contains frequencies up to $f_s/2$. Based on this fact the entire nonlinear processing can be performed without over-sampling and down-sampling by the system shown in Fig. 5.19 [SZ99]. The input signal is split into several lowpass versions which are forwarded to an individual nonlinearity. The output signal is a weighted combination of the individual output signals after passing a nonlinearity. With this approach the problem of aliasing is

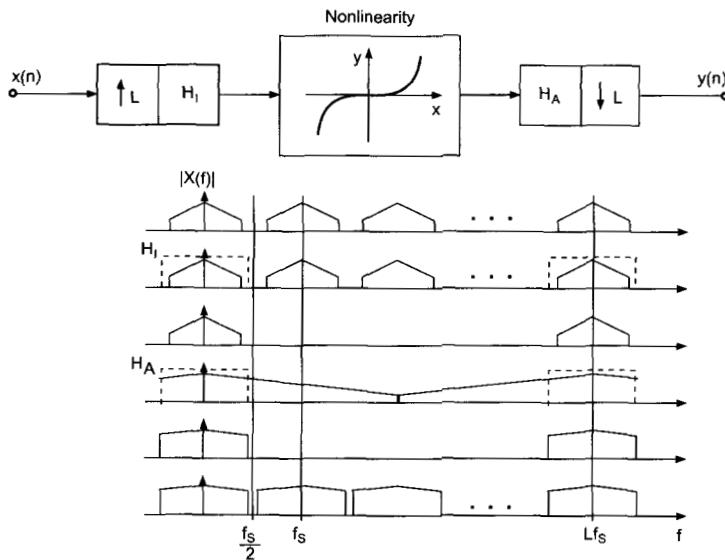


Figure 5.18 Nonlinear processing by over-sampling techniques.

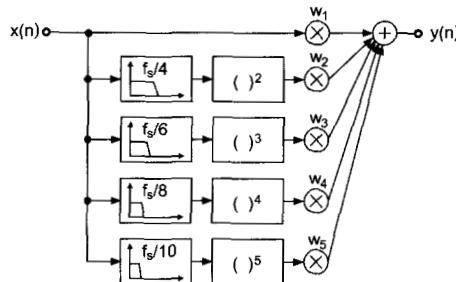


Figure 5.19 Nonlinear processing by band-limiting input range [SZ99].

avoided and an equivalent approximation to the over-sampling technique is achieved. A comparison with our previous discussion on factored Volterra kernels shows also a close connection. As a conclusion for a special static nonlinearity applied to an input signal, the input signal has to be filtered by a lowpass of cut-off frequency $f_s/(2 \cdot \text{order of the Taylor series})$, otherwise aliasing distortions will occur.

5.3.2 Valve Simulation

Valve or tube devices dominated signal processing during the first part of the last century and have experienced a revival in audio processing every decade since their introduction [Bar98, Ham73]. One of the most commonly used effect for electric guitars is the amplifier and especially the valve amplifier. The typical behavior

of the amplifier and the connected loudspeaker cabinet have demonstrated their influence on the sound of rock music over the past decades. Besides the two most important guitars, namely the Fender Stratocaster and the Gibson Les Paul, several valve amplifiers have helped in creating exciting sounds from these classic guitars.

Introduction

Vintage valve amplifiers. An introduction to valve amplifiers and their history can be found in [Fli93, Bar98] where several comments on sound characteristics are published. We will concentrate on the most important amplifier manufacturers over the past and point out some characteristic features.

- **Fender:** The Fender series of guitar amplifiers goes back to the year 1946 when the first amplifiers were introduced. These were based on standard tube schematics supplied by the manufacturers of tubes. Over the years modifications of the standard design approach were integrated in response to musicians' needs and proposals. The range of Fender amplifiers is still expanding but also reissues of the originals are very popular with musicians. The sound of Fender amplifiers is the “classic tube sound”. For more information¹ we refer to [Fli93].
- **Vox:** The sound of the VOX AC30/4 is best characterized by guitar player Brian May in [PD93] where he states “the quality at low levels is broad and crisp and unmistakeably *valve like*, and as the volume is turned up it slides into a pleasant, creamy compression and distortion”. There is always a ringing treble quality through all level settings of the amp. The real “soul of the amp” comes out if you play it at full volume and control your sound with the volume knob of your guitar. The heart of the sound characteristic of the VOX AC30/4 is claimed to be the use of EL84s, NFB and cathode-biasing and Class A configuration. The four small EL84s should sound more lively than the bigger EL34s. The sound of VOX AC30 can be found on recordings by Brian May, Status Quo, Tom Petty and Bryan Adams.
- **Marshall:** The Fender Bassman 5F6 was the basis for the Marshall JTM 45. The differences between both are discussed in [Doy93] and are claimed to be the output transformers, speakers, input valve and feedback circuit, although the main circuit diagrams are nearly identical. The sound of Marshall is characterized by an aggressive and “crunchy” tone with brilliant harmonics, as Eric Clapton says, “I was probably playing full volume to get that sound” [Doy93]. Typical representatives of the early Marshall sound are Jimi Hendrix, Eric Clapton, Jimmy Page and Ritchie Blackmore.
- **Mesa-Boogie:** Two cascaded pre-amp stages and a master volume for the power amp is the basis for Mesa-Boogie amps. An ambassador for Mesa-Boogie sound is Carlos Santana.

¹<http://www.fender.com>, <http://www.ampwares.com>

Signal Processing

The sound of a valve amplifier is based on a combination of several important factors. First of all the main processing features of valves or tubes are important [Bar98, Ham73]. Then the amplifier circuit has its influence on the sound and, last but not least, the chassis and loudspeaker combination play an important role in sound shaping. We will discuss all three factors now.

Valve basics. *Triode* valves [Rat95, RCA59] consisting of three electrodes, namely the anode, cathode and gate, are considered as having a warm and soft sound characteristic. The main reason for this is the nonlinear transfer function for anode current versus input gate voltage of the triode which is shown in Fig. 5.20. This nonlinear curve has a quadratic shape. An input signal represented by the gate voltage U_G delivers an anode output current $I_A = f(U_G)$ representing the output signal. The corresponding output spectrum shows a second harmonic as well as the input frequency. This second harmonic can be lowered in amplitude when the operating point of the nonlinear curve is shifted right and the input voltage is applied to the more linear region of the quadratic curve. The dc component in

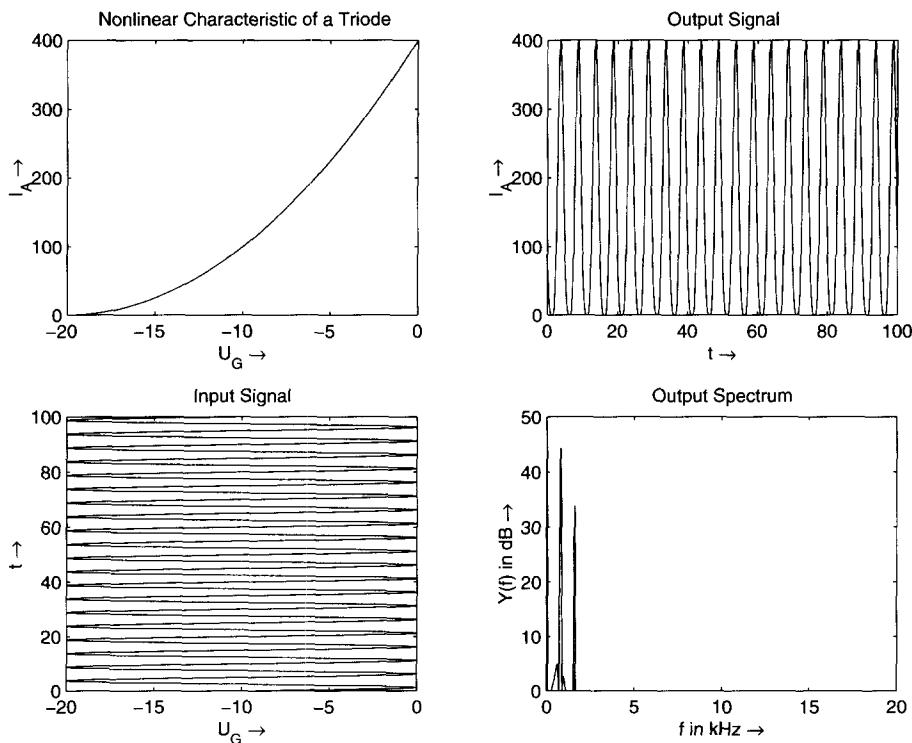


Figure 5.20 Triode: nonlinear characteristic curve $I_A = f(U_G)$ and nonlinear effect on input signal. The output spectrum consists of the fundamental input frequency and a second harmonic generated by the quadratic curve of the triode.

the output signal can be suppressed by a subsequent highpass filter. Note also the asymmetrical soft clipping of the negative halves of the output sinusoid, which is the result of the quadratic curve of the triode. Input stages of valve amplifiers make use of these triode valves. A design parameter is the operating point which controls the amplitude of the second harmonic.

Pentode valves, which consist of five electrodes, are mainly used in the power amp stage [Rat95, RCA59]. They have a static characteristic like an S-curve shown in Fig. 5.21 which shows the anode output current I_A versus the input gate voltage U_G . If the entire static characteristic curve is used, the output signals is compressed for higher input amplitudes leading to a symmetrical soft clipping. The corresponding output spectrum shows the creation of odd order harmonics. For lower input amplitudes the static characteristic curve operates in a nearly linear region, which again shows the control of the nonlinear behavior by properly selecting the operating point. Several static characteristic curves can be chosen to achieve the S-like curve of pentodes.

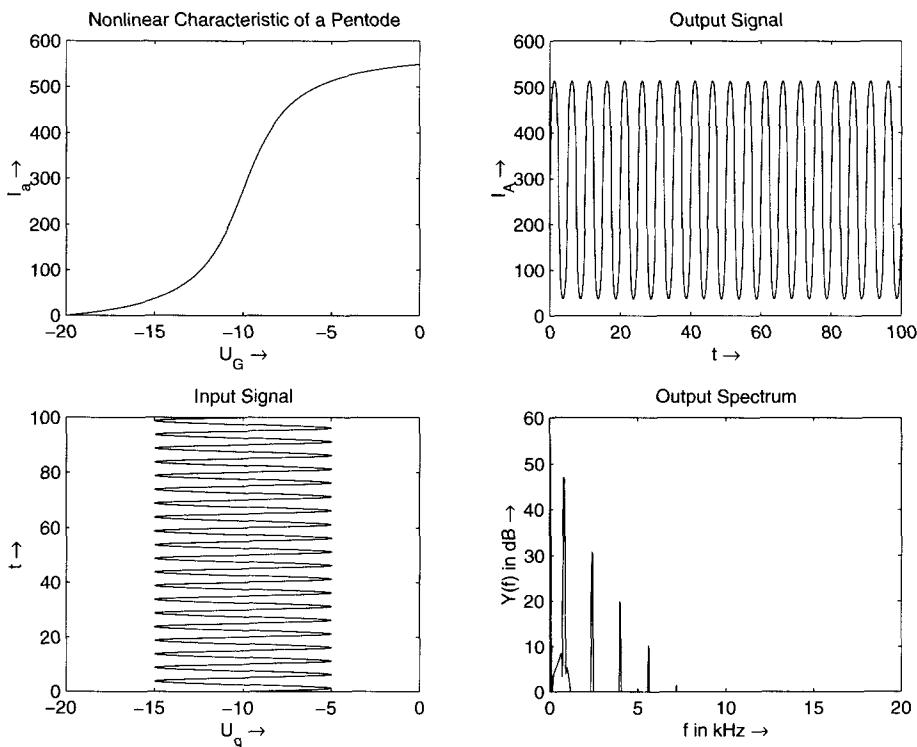


Figure 5.21 Pentode: nonlinear characteristic curve $I_A = f(U_G)$ and nonlinear effect on input signal.

The technical parameters of valves have wide variation, which leads to a wide variation of sound features, although selected valves with small changes of para-

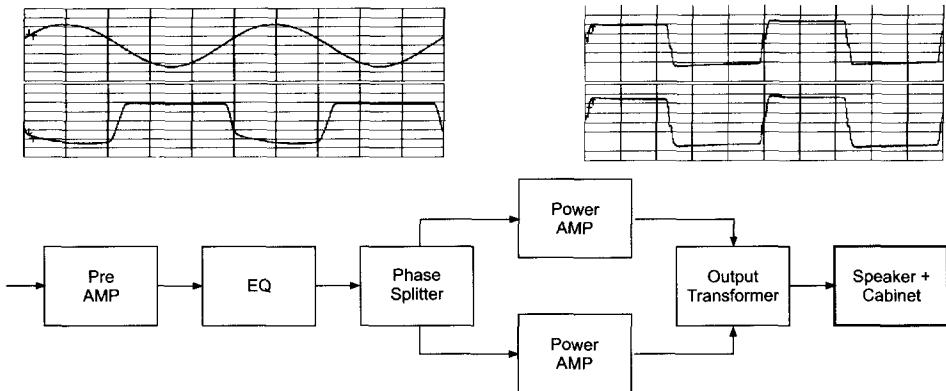


Figure 5.22 Main stages of a valve amplifier. Upper left plot shows signal after pre-amplifier, lower left plot shows signal after phase splitter, upper right plot shows signal after power amplifier and lower right plot shows signal after output transformer.

meters are of course available. All surrounding environmental parameters like humidity and temperature have their influence as well.

Valve amplifier circuits. Valve amplifier circuits are based on the block diagram in Fig. 5.22. Several measured signals from a VOXAC30 at different stages of the signal flow path are also displayed. This will give an indication of typical signal distortions in valve amplifiers. The main stages of a valve amplifier are given below:

- the *input stage* consists of a triode circuit providing the input matching followed by a volume control for the next stages.
- the *tone control* circuitry is based on low/high frequency shelving filters.
- the *phase inversion/splitting* stage for providing symmetrical power amp feeding. This phase splitter delivers the original input for the upper power amp and a phase inverted replica of the input for the lower power amp.
- the *power amp* stage performs individual amplification of the original and the phase inverted replica in a class A, class B or class C configuration (see Fig. 5.23). Class A is shown in the upper left plot, where the output signal is valid all the time. Class B performs amplification only for one half wave and class C only for a portion of one half wave. Class A and class AB (see lower part of Fig. 5.23) are the main configurations for guitar power amplifiers. For class AB operation the working point lies in between class A and class B. The signals after amplification in both parts of the power amplifier are shown.
- the *output transformer*, shown in Fig. 5.24, performs the subtraction of both waveforms delivered by the power amplifiers which leads to the output signal $I_{A1} - I_{A2}$ as shown in Fig. 5.23. The nonlinear behavior of transformers is beyond the scope of this discussion.

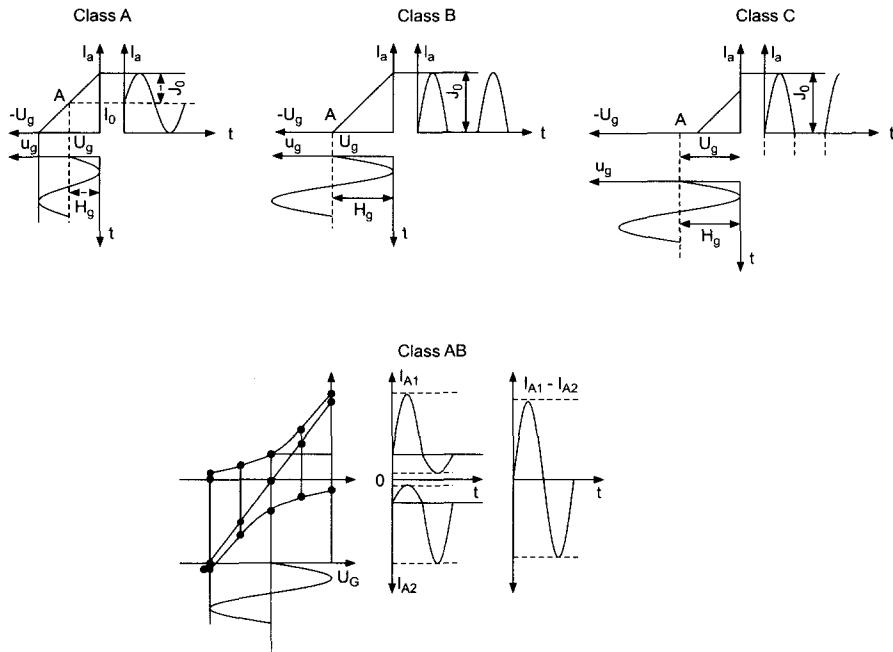


Figure 5.23 Power amplifier operation (upper plots: left class A, middle class B, right class C, lower plot: class AB operation).

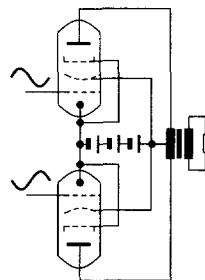


Figure 5.24 Power amplifier stage and output transformer.

- the chassis and the loudspeakers are arranged in several combinations ranging from 2x12 to 4x10 in closed or open cabinets. Simulations of these components can be done by impulse response measurements of the loudspeaker and cabinet combination.

As well as the discussed topics the influence of the power supply with valve rectifier (pp. 51-54 in [vdL97]) is claimed to be of importance. A soft reduction of the power supply voltage occurs when in a high power operation short transients need a high current. This power supply effect leads to a soft clipping of the audio signal.

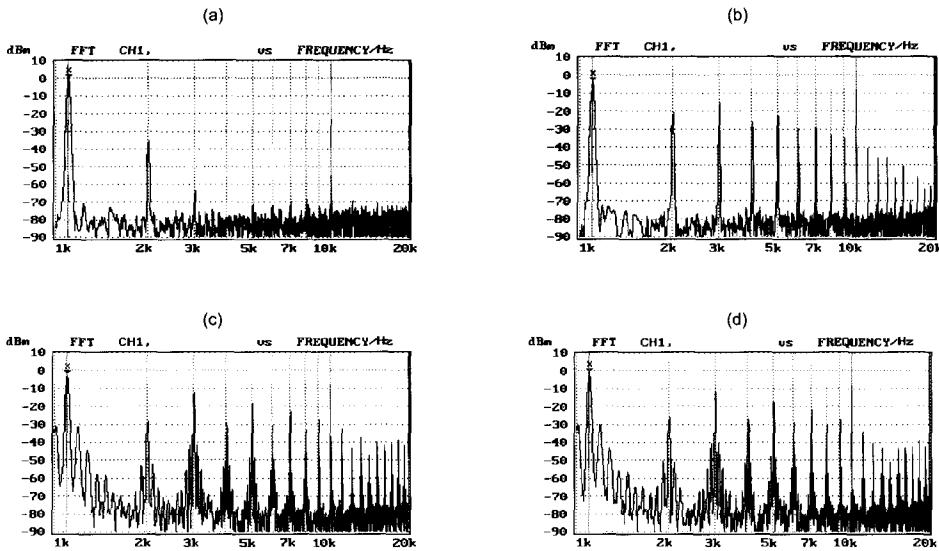


Figure 5.25 VOX AC30/4 spectra at different stages: (a) input stage, (b) output phase splitter, (c) output power amp and (d) output of transformer.

The corresponding spectra of the signals for the VOX AC30/4 measurements are shown in Fig. 5.25. The distortion characteristic can be visualized by the waterfall representation of short-time FFTs for a chirp input signal in Fig. 5.26. The individual distortion components for the second up to the fifth harmonic are shown in Fig. 5.27 for each signal in the VOX AC30/4 amplifier.

Musical Applications

Musical applications of valve amplifiers can be found on nearly every recording featuring guitar tracks. Ambassadors of innovative guitar players from the blues to early rock period are B. B. King, Albert King and Chuck Berry, who mainly used valve amplifiers for their warm and soft sound. Representatives of the classic rock period are Jimi Hendrix [m-Hen67a, m-Hen67b, m-Hen68], Eric Clapton [m-Cla67], Jimmy Page [m-Pag69], Ritchie Blackmore [m-Bla70], Jeff Beck [m-Bec89] and Carlos Santana [m-San99]. All make extensive use of valve amplification and special guitar effect units. There are also players from the new classic period like Eddie van Halen, Steve Ray Vaughan and Steve Morse up to the new guitar heroes such as Steve Lukather, Joe Satriani, Gary Moore, Steve Vai and Paul Gilbert, who are using effect devices together with valve amplifiers. New guitar amplifier designs with digital preamplifiers are based on digital modeling technology,² where a modeling of classic valve amplifiers is performed together with new valve-based algorithms for guitar and bass guitar processing.

²<http://www.line6.com>

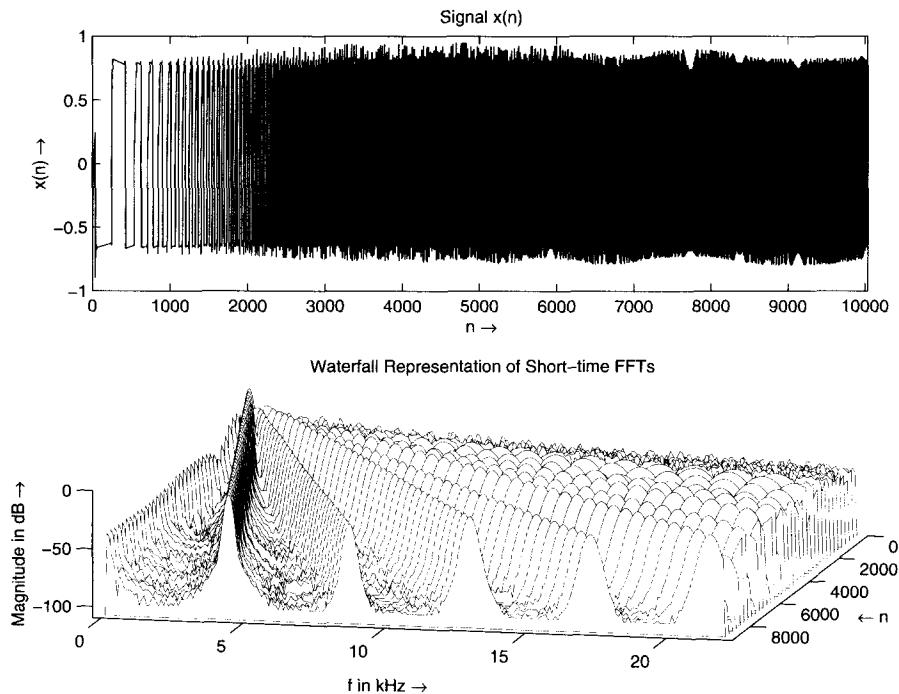


Figure 5.26 Short-time FFTs (waterfall representation) of VOXAC30 with a chirp input signal. The amplifier is operating with full volume setting.

Valve microphones, preamplifiers and effect devices such as compressors, limiters and equalizers are also used for vocal recordings where the warm and subtle effect of valve compression is applied. A lot of vocalists prefer recording with valve condenser microphones because of their warm low end and smooth top end frequency response. Also the recording of acoustical instruments such as acoustic guitars, brass instruments and drums benefit from being processed with valve outboard devices. Valve processors also assist the mixing process for individual track enhancing and on the mix buses. The demand for valve outboard effects and classic mixing consoles used in combination with digital audio workstations has led back to entire valve technology mixing consoles. For the variety of digital audio workstations a lot of plug-in software modules for valve processing are available on the marketplace.

5.3.3 Overdrive, Distortion and Fuzz

Introduction

As pointed out in the section on valve simulation, the distorted electric guitar is a central part of rock music. As well as the guitar amplifier as a major sound effect device, several stomp boxes (foot-operated pedals) have been used by guitar play-

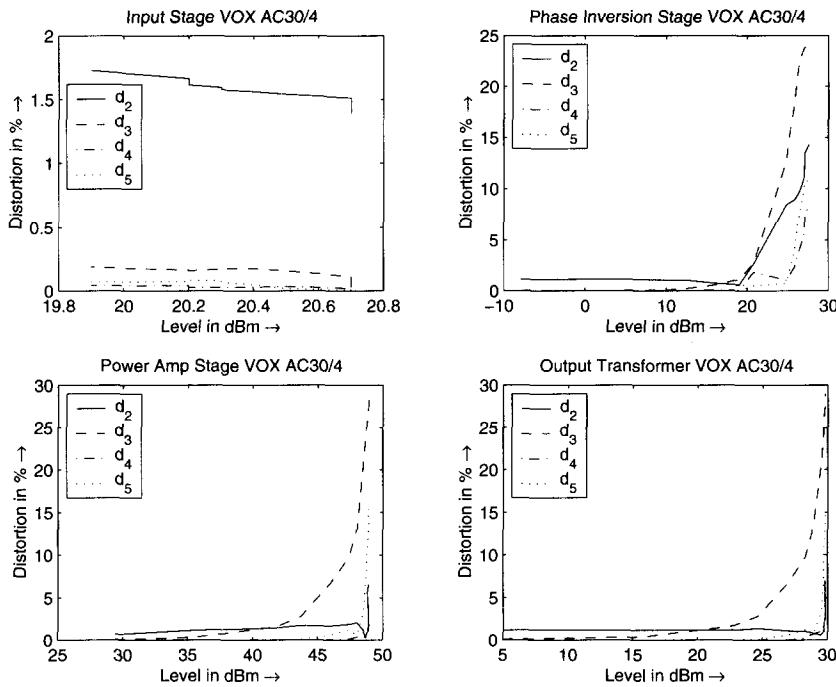


Figure 5.27 VOX AC30 - Distortion components versus signal level at different stages.

ers for the creation of their typical guitar sound. Guitar heroes like Jimi Hendrix have made use of several small analog effect devices to achieve their unmistakable sound. Most of these effect devices have been used to create higher harmonics for the guitar sound in a faster way and at a much lower sound level compared to valve amplifiers. In this context terms like overdrive, distortion, fuzz and buzz are used. Several definitions of overdrive, distortion and fuzz for musical applications especially in the guitar player world are available.³ For our discussion we will define *overdrive* as a first state where a nearly linear audio effect device at low input levels is driven by higher input levels into the nonlinear region of its characteristic curve. The operating region is in the linear region as well as in the nonlinear region with a smooth transition. The main sound characteristic is of course from the nonlinear part. Overdrive has a warm and smooth sound. The second state is termed *distortion*, where the effects device mainly operates in the nonlinear region of the characteristic curve and reaches the upper input level, where the output level is fixed to a maximum level. Distortion covers a wide tonal area starting beyond tube warmth to buzz saw effects. All metal and grunge sounds fall into this category. The operating status of *fuzz* is represented by a completely nonlinear behavior of the effect device with a sound characterized by the guitar player terms “harder” and “harsher” than distortion. Fuzz also means not clear, distinct, or precise and

³A Musical Distortion Primer by R.G. Keen on <http://www.geofec.com>

unpredictable. The fuzz effect is generally used on single-note lead lines.

Signal Processing

Symmetrical soft clipping. For overdrive simulations a symmetrical soft clipping of the input values has to be performed. One possible approach for a soft saturation nonlinearity [Sch80] is given by

$$f(x) = \begin{cases} 2x & \text{for } 0 \leq x \leq 1/3 \\ \frac{3-(2-3x)^2}{3} & \text{for } 1/3 \leq x \leq 2/3 \\ 1 & \text{for } 2/3 \leq x \leq 1. \end{cases} \quad (5.7)$$

The static input to output relation is shown in Fig. 5.28. Up to the threshold of 1/3 the input is multiplied by two and the characteristic curve is in its linear region. Between input values of 1/3 up to 2/3, the characteristic curve produces a soft compression described by the middle term of equation (5.7). Above input values of 2/3 the output value is set to one. The corresponding M-file 5.5 for overdrive with symmetrical soft clipping is shown next.

```
M-file 5.5 (symclip.m)
function y=symclip(x)
% y=symclip(x)
% "Overdrive" simulation with symmetrical clipping
% x - input
N=length(x);
th=1/3; % threshold for symmetrical soft clipping
           % by Schetzen Formula
for i=1:1:N,
    if abs(x(i))< th, y(i)=2*x(i);end;
    if abs(x(i))>=th,
        if x(i)> 0, y(i)=(3-(2-x(i)*3).^2)/3; end;
        if x(i)< 0, y(i)=-(3-(2-abs(x(i))*3).^2)/3; end;
    end;
    if abs(x(i))>2*th,
        if x(i)> 0, y(i)=1;end;
        if x(i)< 0, y(i)=-1;end;
    end;
end;
```

Figure 5.29 shows the waveforms of a simulation with the above described characteristic curve and a sinusoid of 1 kHz. In the upper left plot the output signal from sample 0 up to 250 is shown which corresponds to the saturated part of the characteristic curve. The tops and bottoms of the sinusoid run with a soft curve towards the saturated maximum values. The upper right plot shows the output signal from sample 2000 up to 2250, where the maximum values are in the soft clipping region of the characteristic curve. Both the negative and the positive top of the sinusoid are

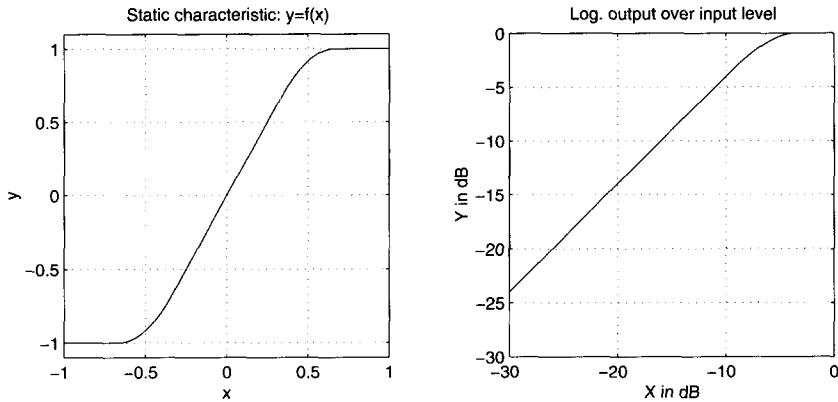


Figure 5.28 Static characteristic curve of symmetrical soft clipping (right part shows logarithmic output value versus input value).

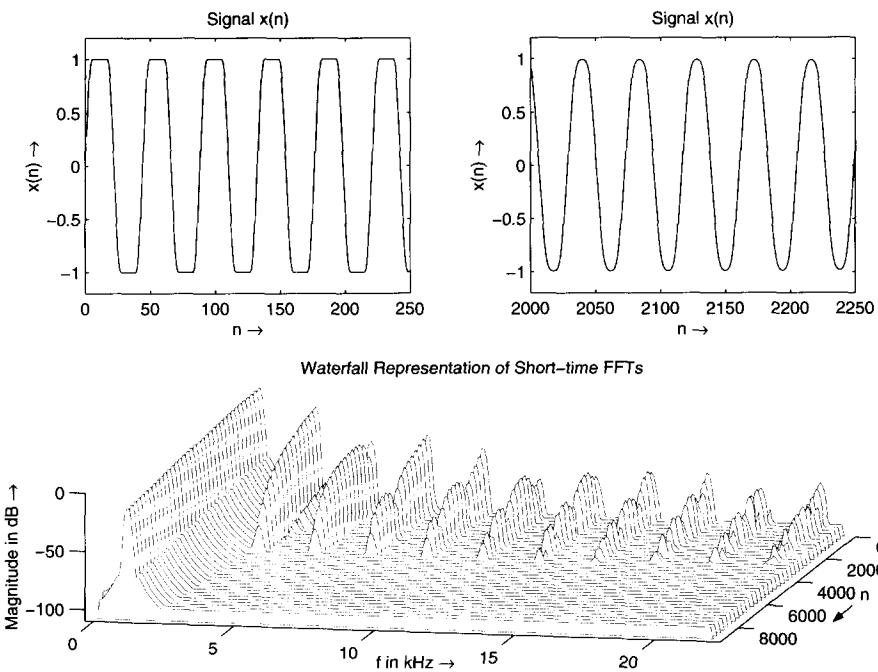


Figure 5.29 Short-time FFTs (waterfall representation) of symmetrical soft clipping for a decaying sinusoid of 1 kHz.

rounded in their shape. The lower waterfall representation shows the entire decay of the sinusoid down to -12 dB. Notice the odd order harmonics produced by this nonlinear symmetrical characteristic curve, which appear in the nonlinear region of the

characteristic curve and disappear as soon as the lower threshold of the soft compression is reached. The prominent harmonics are the third and the fifth harmonic. The slow increase or decrease of higher harmonics is the major property of symmetrical soft clipping. As soon as simple hard clipping without a soft compression is performed, higher harmonics appear with significant higher levels (see Fig. 5.30). The discussion of overdrive and distortion has so far only considered the creation of harmonics for a single sinusoid as the input signal. Since a single guitar tone itself consists of the fundamental frequency plus all odd and even order harmonics, the sum of sinusoids are always processed by a nonlinearity. The nonlinearity also produces sum and difference frequencies, which sound very disturbing. The control of these sum and difference frequencies goes beyond the current treatment of the subject.

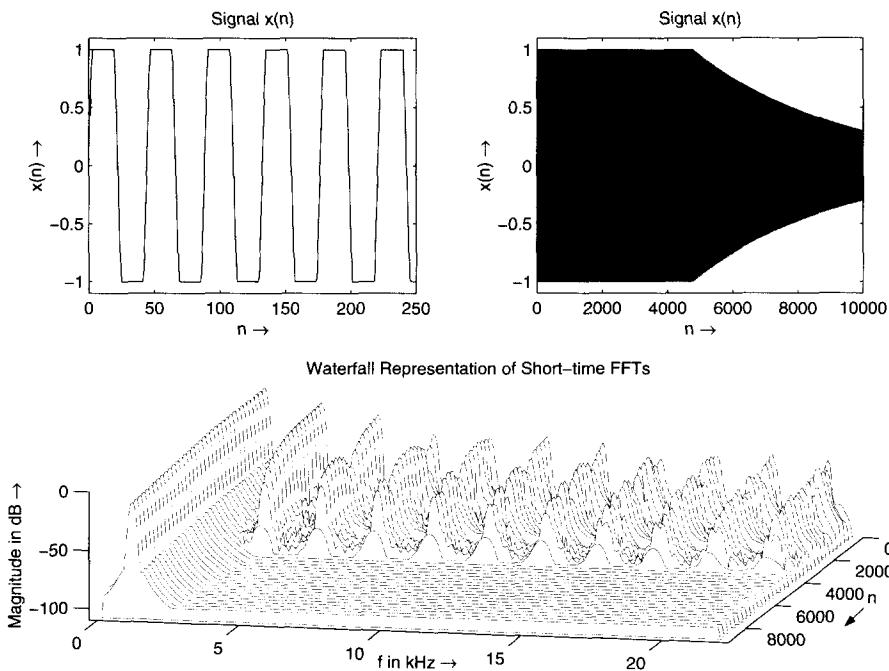


Figure 5.30 Short-time FFTs (waterfall representation) of symmetrical hard clipping for a decaying sinusoid of 1 kHz.

Asymmetrical clipping. We have already discussed the behavior of triode valves in the previous section which produce an asymmetrical overdrive. One famous representative of asymmetrical clipping is the *Fuzz Faze* which was used by Jimi Hendrix. The basic analog circuit is shown in Fig. 5.31⁴ and consists only of a few components with two transistors in a feedback arrangement. The output signals for various input levels are presented in Fig. 5.32 in conjunction with the corresponding spectra for

⁴The Technology of the Fuzz Face by R.G. Keen on <http://www.geofex.com>

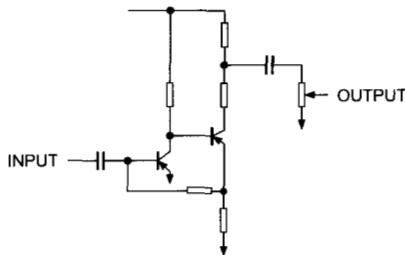


Figure 5.31 Analog circuit of Fuzz Face.

a 1 kHz sinusoid. The upper plots down to the lower plots show an increasing input

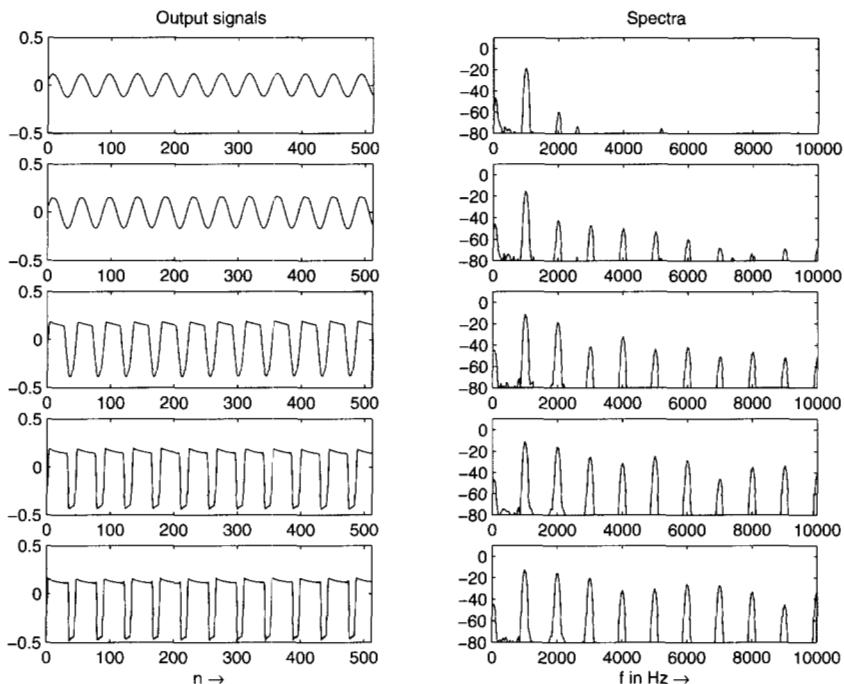


Figure 5.32 Signals and corresponding spectra of Fuzz Face.

level. For low level input signals the typical second harmonic of a triode valve can be noticed although the time signal shows no distortion components. With increasing input level the second harmonic and all even order harmonics as well as odd order harmonics appear. The asymmetrical clipping produces enhanced even order harmonics as shown in the third row of Fig. 5.32. Notice that only the top of the positive maximum values are clipped. As soon as the input level further increases, the negative part of the waveform is clipped. The negative clipping level is lower than the positive clipping value and so asymmetrical clipping is performed. When

both positive and negative clipping levels are exceeded, the odd order harmonics come up but the even order harmonics are still present.

Short-time Fourier transforms (in waterfall representation) for an increasing 1 kHz sinusoid together with two waveforms are shown in Fig. 5.33.

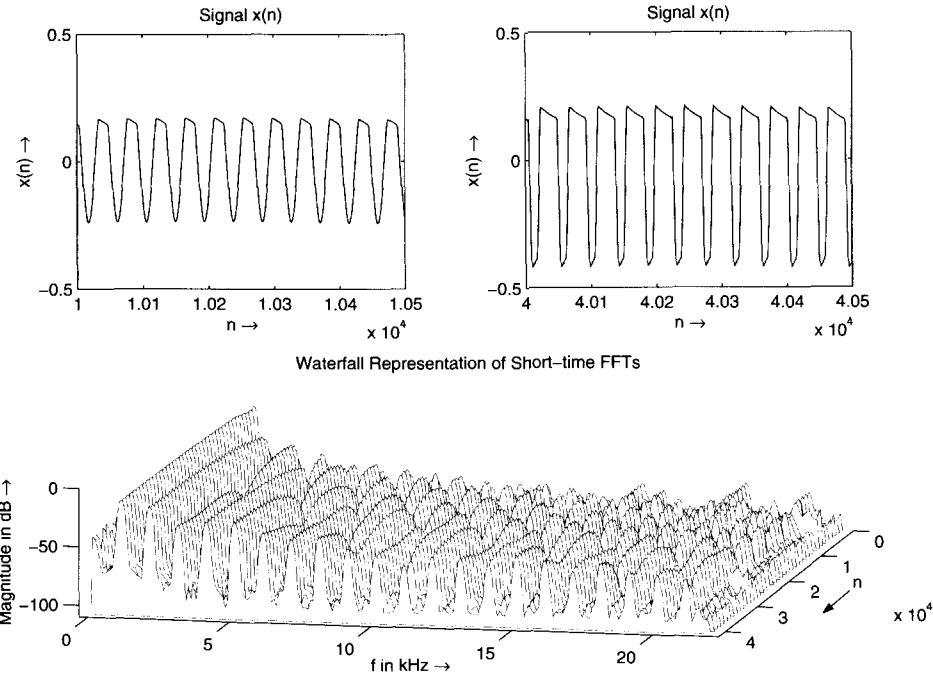


Figure 5.33 Short-time FFTs (waterfall representation) of Fuzz Face for an increasing 1 kHz sinusoid. The upper plots show segments of samples from the complete analysed signal.

A proposal for asymmetrical clipping [Ben97] used for tube simulation is given by

$$f(x) = \frac{x - Q}{1 - e^{-\text{dist}(x-Q)}} + \frac{Q}{1 - e^{\text{dist}(Q)}}, \quad Q \neq 0, x \neq Q. \quad (5.8)$$

The underlying design parameters for the simulation of tube distortion are based on the mathematical model [Ben97] where no distortion should occur when the input level is low (the derivative of $f(x)$ has to be $f'(0) \approx 1$ and $f(0) = 0$). The static characteristic curve should perform clipping and limiting of large negative input values and approximately linear for positive values. The result of equation (5.8) is shown in Fig. 5.34.

The following M-file 5.6 performs equation (5.8) from [Ben97]. To remove the dc component and to shape higher harmonics, additional lowpass and highpass filtering of the output signal is performed.

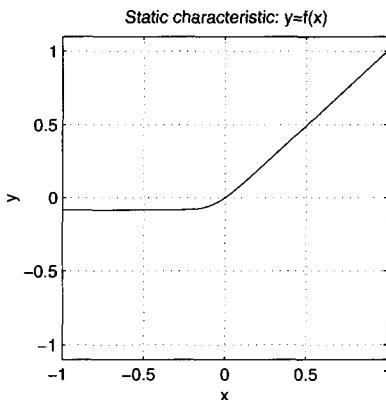


Figure 5.34 Static characteristic curve of asymmetric soft clipping for tube simulation $Q = -0.2$ and $dist = 8$.

M-file 5.6 (tube.m)

```

function y=tube(x, gain, Q, dist, rh, rl, mix)
% y=tube(x, gain, Q, dist, rh, rl, mix)
% "Tube distortion" simulation, asymmetrical function
% x      - input
% gain   - the amount of distortion, >0-
% Q      - work point. Controls the linearity of the transfer
%          function for low input levels, more negative=more linear
% dist   - controls the distortion's character, a higher number gives
%          a harder distortion, >0
% rh    - abs(rh)<1, but close to 1. Placement of poles in the HP
%          filter which removes the DC component
% rl    - 0<rl<1. The pole placement in the LP filter used to
%          simulate capacitances in a tube amplifier
% mix   - mix of original and distorted sound, 1=only distorted
q=x*gain/max(abs(x)); %Normalization
if Q==0
    z=q./(1-exp(-dist*q));
    for i=1:length(q) %Test because of the
        if q(i)==Q %transfer function's
            z(i)=1/dist; %0/0 value in Q
        end;
    end;
else
    z=(q-Q)./(1-exp(-dist*(q-Q)))+Q/(1-exp(dist*Q));
    for i=1:length(q) %Test because of the
        if q(i)==Q %transfer function's
            z(i)=1/dist+Q/(1-exp(dist*Q)); %0/0 value in Q
        end;
    end;
end;

```

```

end;
end;
y=mix*z*max(abs(x))/max(abs(z))+(1-mix)*x;
y=y*max(abs(x))/max(abs(y));
y=filter([1 -2 1],[1 -2*rh rh^2],y); %HP filter
y=filter([1-r1],[1 -r1],y);      %LP filter

```

Short-time FFTs (waterfall representation) of this algorithm applied to a 1 kHz sinusoid are shown in Fig. 5.35. The waterfall representation shows strong even order harmonics and also odd order harmonics.

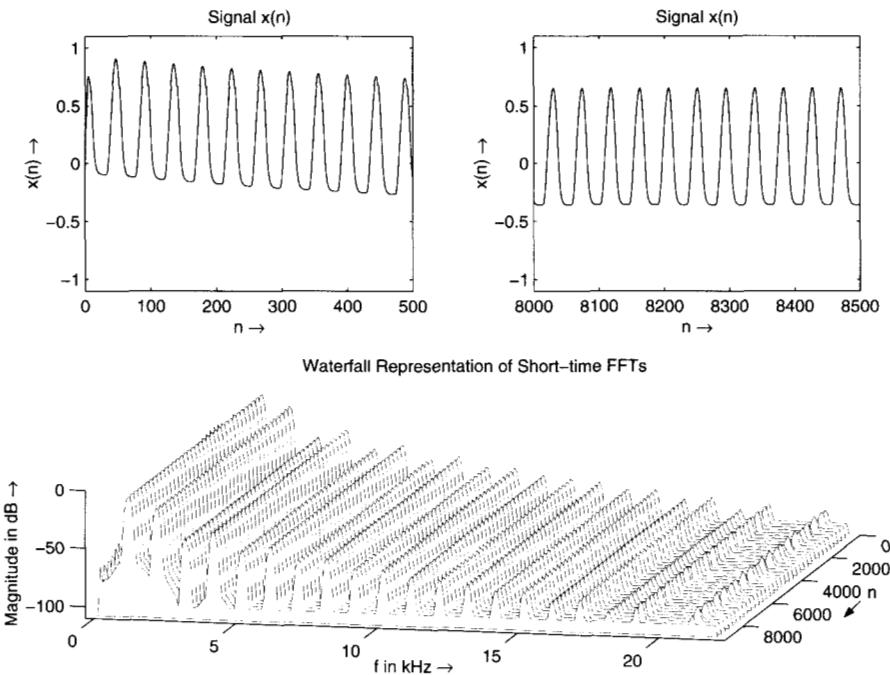


Figure 5.35 Short-time FFTs (waterfall representation) of asymmetrical soft clipping.

Distortion. A nonlinearity suitable for the simulation of distortion [Ben97] is given by

$$f(x) = \frac{x}{|x|} \left(1 - e^{x^2/|x|}\right). \quad (5.9)$$

The M-file 5.7 for performing equation (5.9) is shown next.

M-file 5.7 (fuzzexp.m)

```

function y=fuzzexp(x, gain, mix)
% y=fuzzexp(x, gain, mix)

```

```
% Distortion based on an exponential function
% x      - input
% gain - amount of distortion, >0->
% mix   - mix of original and distorted sound, 1=only distorted
q=x*gain/max(abs(x));
z=sign(-q).*(1-exp(sign(-q).*q));
y=mix*z*max(abs(x))/max(abs(z))+(1-mix)*x;
y=y*max(abs(x))/max(abs(y));
```

The static characteristic curve is illustrated in Fig. 5.36 and short-time FFTs of a decaying 1 kHz sinusoid are shown in Fig. 5.37.

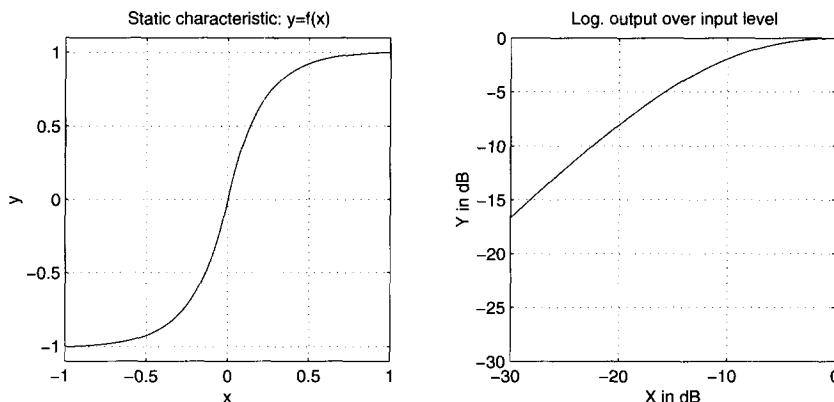


Figure 5.36 Static characteristic curve of exponential distortion.

Musical Applications

There are a lot of commercial stomp effects for guitarists on the market place. Some of the most interesting distortion devices for guitars are the *Fuzz Face* which performs asymmetrical clipping towards symmetrical soft clipping and the *Tube Screamer*,⁵ which performs symmetrical soft clipping.⁶ The *Fuzz Face* was used by Jimi Hendrix and the *Tube Screamer* by Stevie Ray Vaughan. They both offer classical distortion and are well known because of their famous users. It is impossible to explain the sound of a distortion unit without listening personally to it. The technical specifications for the sound of distortion are missing, so the only way to choose a distortion effect is by a comparative listening test.

⁵The Technology of the Tube Screamer by R.G. Keen on <http://www.geofex.com>

⁶GM Arts Homepage <http://www.chariot.net.au/~gmarts/ampovdrv.htm>

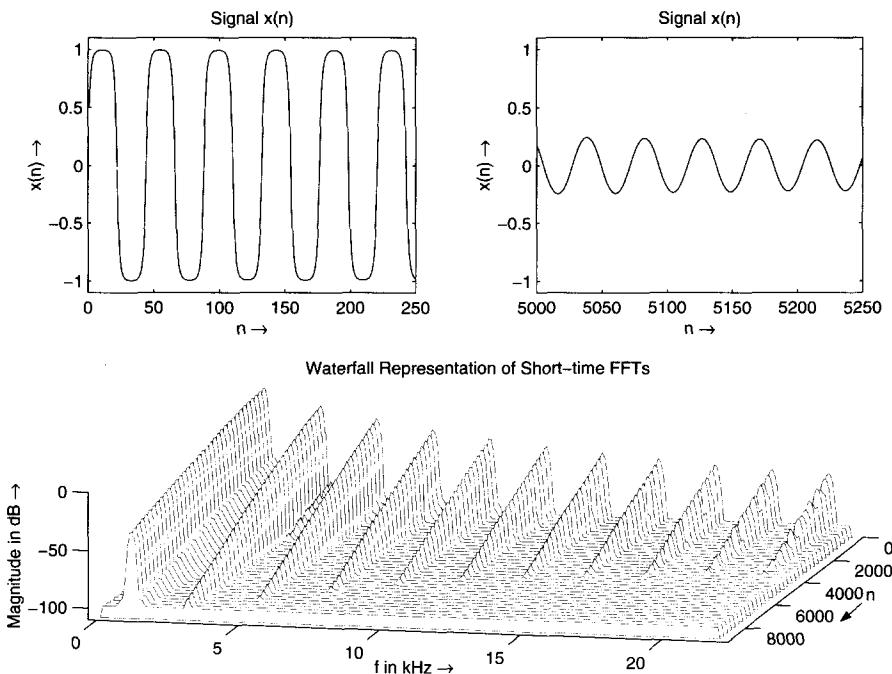


Figure 5.37 Short-time FFTs (waterfall representation) of exponential distortion.

5.3.4 Harmonic and Subharmonic Generation

Introduction

Harmonic and subharmonic generation are performed by simple analog or digital effect devices which should produce an octave above and/or an octave below a single note. Advanced techniques to achieve pitch shifting of instrument sounds will be introduced in Chapter 7. Here, we will focus on simple techniques, which lead to the generation of harmonics and subharmonics.

Signal Processing

The signal processing algorithms for harmonic and subharmonic generation are based on simple mathematical operations like absolute value computation and counting of zero crossings, as shown in Fig. 5.38 when an input sinusoid has to be processed (first row shows time signal and corresponding spectrum).

The second row of Fig. 5.38 demonstrates half-wave rectification, where positive values are kept and negative values are set to zero. This operation leads to the generation of even order harmonics. Full-wave rectification, where the absolute value is taken from the input sequence, leads to even order harmonics as shown in the third row of Fig. 5.38. Notice the absence of the fundamental frequency. If a zero crossing

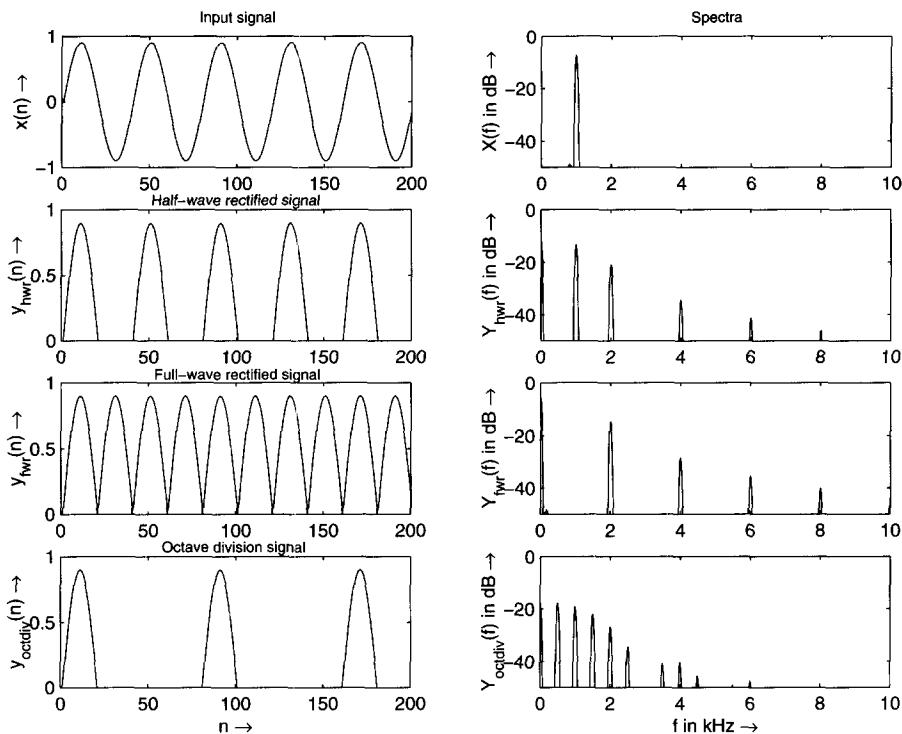


Figure 5.38 Signals and corresponding spectra of halve-wave rectification, full-wave rectification and octave division.

counter is applied to the half-wave or the full-wave rectified signal, a predefined number of positive wave parts can be set to zero to achieve the signal in the last row of Fig. 5.38. This signal has a fundamental frequency which is one octave lower than the input frequency in the first row of the figure, but also shows harmonics of this new fundamental frequency. If appropriate lowpass filtering is applied to such a signal, only the fundamental frequency can be obtained which is then added to the original input signal.

Musical Applications

Harmonic and subharmonic generation is mostly used on single note lead lines, where an additional harmonic or subharmonic frequency helps to enhance the octave effect. Harmonic generators can be found in stomp boxes for guitar or bass guitar and appear under the name octaver. Subharmonic generation is often used for solo and bass instruments to give them an extra bass boost or simply a fuzz bass character.

5.3.5 Tape Saturation

Introduction and Musical Application

The special sound characteristic of analog tape recordings has been acknowledged by a variety of producers and musicians in the field of rock music. They prefer doing multi-track recordings with analog tape-based machines and use the special physics of magnetic tape recording as an analog effects processor for sound design. One reason for their preference for analog recording is the fact that magnetic tape goes into distortion gradually [Ear76] (pp. 216-218) and produces those kinds of harmonics which help special sound effects on drums, guitars and vocals.

Signal Processing

Tape saturation can be simulated by the already introduced techniques for valve simulation. An input level derived weighting curve is used for generating a gain factor which is used to compress the input signal. A variety of measurements of tape recordings can help in the design of such processing devices. An example of the input/output behavior is shown in Fig. 5.39 and a short-time FFT of a sinusoid input signal in Fig. 5.40 illustrates a tape saturation algorithm. For low-level inputs the transfer characteristic is linear without any distortions. A smooth soft compression simulates the gradually increasing distortion of magnetic tape.

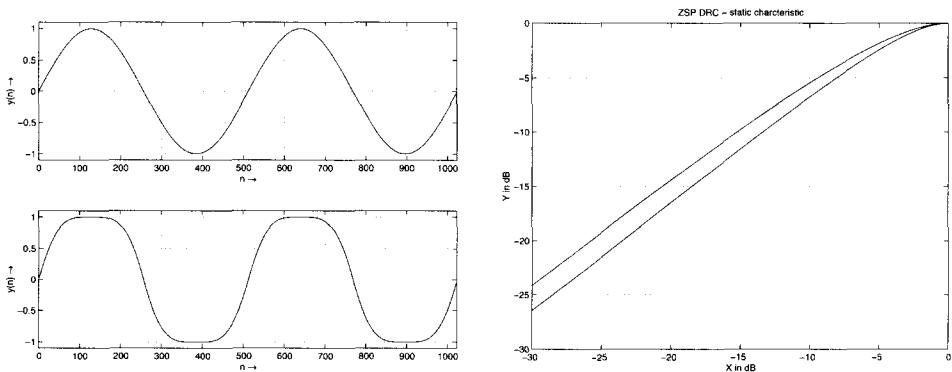


Figure 5.39 Tape saturation: input and output signal (left) and static characteristic curve.

5.4 Exciters and Enhancers

5.4.1 Exciters

Introduction

An exciter is a signal processor that emphasizes or de-emphasizes certain frequencies in order to change a signal's timbre. An exciter increases brightness without

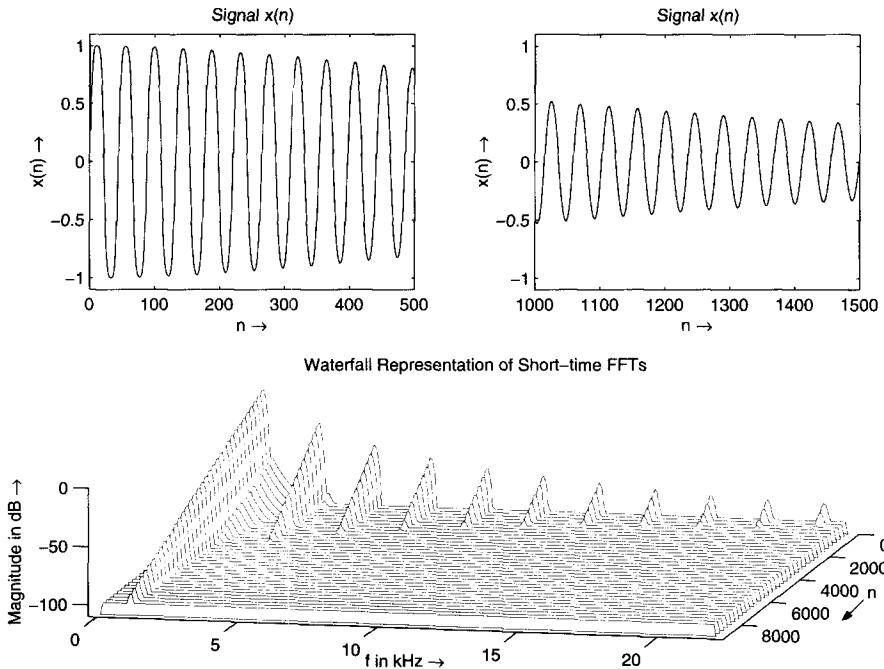


Figure 5.40 Tape saturation: short-time FFTs (waterfall representation) for decaying sinusoid of 1 kHz.

necessarily adding equalization. The result is a brighter, “airier” sound without the stridency that can sometimes occur by simply boosting the treble. This is often accomplished with subtle amounts of high-frequency distortion, and sometimes, by playing around with phase shifting. Usually there will only be one or two parameters, such as exciter mix and exciter frequency. The former determines how much “excited” sound gets added to the straight sound, and the latter determines the frequency at which the exciter effect starts [Whi93, And95, Dic87, WG94].

This effect was discovered by the Aphex company and “Aural Exciter” is a trademark of this company. The medium and treble parts of the original signal are processed by a nonlinear circuit that generates higher overtones. These components are then mixed to some extent to the original signal. A compressor at the output of the nonlinear element makes the effect dependent on the input signal. The initial part of percussive sounds will be more enriched than the following part, when the compressor limits the effect depth. The enhanced imaging or spaciousness is probably the result of the phase rotation within the filter [Alt90].

Signal Processing

Measurement results of the APHEX Aural Exciter are shown in Fig. 5.41 and in Fig. 5.42 where the generation of a second harmonic is clearly visible. The input

signal is a chirp signal with an increasing frequency up to 5 kHz. Signal processing techniques to achieve the effect have already been discussed in the previous sections. The effect is created in the side chain path and is mixed with the input signal.

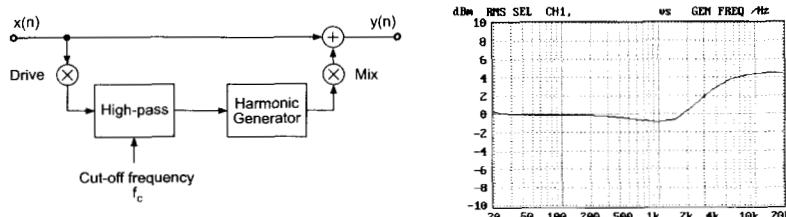


Figure 5.41 Block diagram of the psycho-acoustic equalizer APHEX Aural Exciter and frequency response.

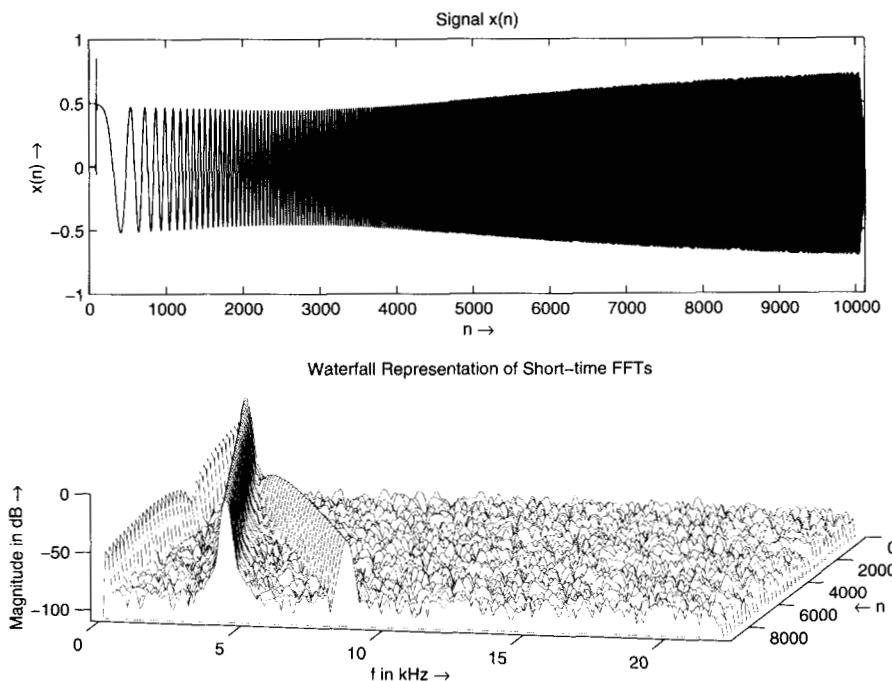


Figure 5.42 Short-time FFTs (waterfall representation) of a psycho-acoustic equalizer.

Musical Applications

The applications of this effect are widespread and range from single instrument enhancement to enhancement of mix buses and stereo signals. The effect increases

the presence and clarity of a single instrument inside a mix and helps to add natural brightness to stereo signals. Applied to vocals and speech the effect increases intelligibility. Compared to equalizers the sound level is only increased slightly. The application of this effect only makes sense if the input signal lacks high frequency contents.

5.4.2 Enhancers

Introduction

Enhancers are signal processors which combine equalization together with nonlinear processing. They perform equalization according to the fundamentals of psycho-acoustics [ZF90] and introduce a small amount of distortion in a just noticeable manner. An introduction to the ear's own nonlinear distortions, sharpness, sensory pleasantness and roughness can be also be found in [ZF90]. A lot of recording engineers and musicians listen under slightly distorted conditions during recording and mixing sessions.

Signal Processing

As an example of this class of devices the block diagram and the frequency response of the SPL vitalizer are shown in Fig. 5.43. This effect processor has also a side chain path which performs equalization with a strong bass enhancement, a mid-frequency cut and a high-frequency boost. The short-time FFT of the output signal when a chirp input signal is applied is shown in Fig. 5.44. The resulting waterfall representation clearly shows higher harmonics generated by this effect processor.

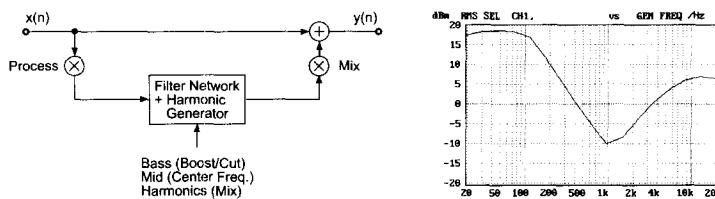


Figure 5.43 Block diagram of the psycho-acoustic equalizer SPL Vitalizer and frequency response.

Further refinements of enhancers can be achieved through multiband enhancers which split the input signal into several frequency bands. Inside each frequency band nonlinear processing plus filtering is performed. The output signals of each frequency band are weighted and summed up to form the output signal (see Fig. 5.45).

Musical Applications

The main applications of such effects are single track processing as a substitute for the equalizers inside the input channels of mixing consoles and processing of final

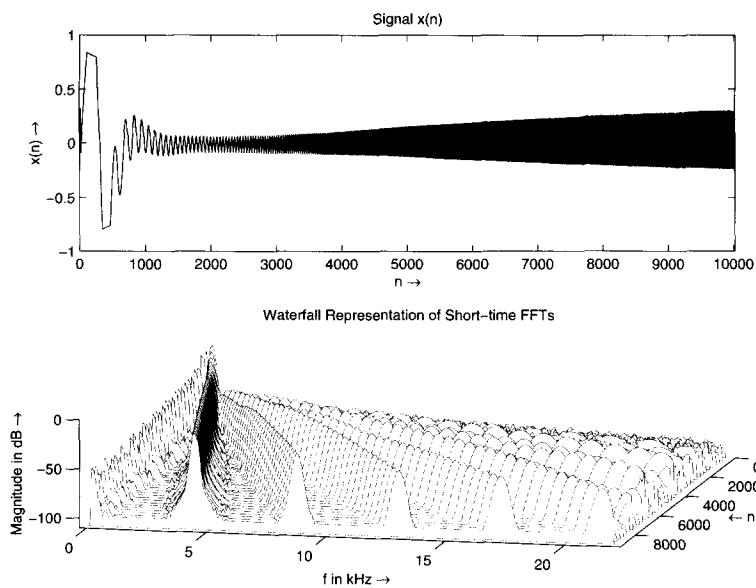


Figure 5.44 Short-time FFTs (waterfall representation) of psycho-acoustic equalizer SPL Vitalizer.

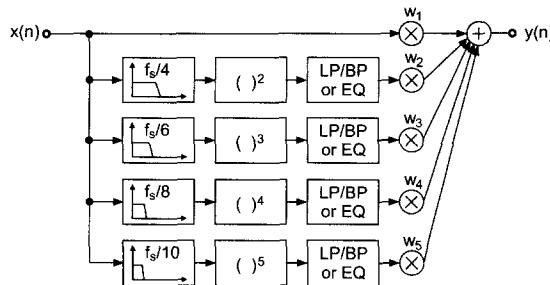


Figure 5.45 Multiband enhancer with nonlinear processing in frequency bands.

mixes. The side chain processing allows the subtle mix of the effects signal together with the input signal. Further applications are stereo enhancement for broadcast stations and sound reinforcement.

5.5 Conclusion

The most challenging tools for musicians and sound engineers are nonlinear processors such as dynamics processors, valve simulators and excitors. The successful application of these audio processors depends on the appropriate control of these devices. A variety of interactive control parameters influences the resulting sound

quality.

The primary purpose of this chapter is to enable the reader to attain a fundamental understanding of different types of nonlinear processors and the special properties of nonlinear operations applied to the audio signal. Dynamics processors need a careful consideration of the interaction of thresholds and time-constants to achieve sonic purity and avoid aliasing distortion. On the other hand nonlinear processors are used for the simulation of valve amplifiers or nonlinear audio systems, where a special kind of nonlinearity provides a sound distortion with accepted sound characteristic. We have presented the basics of nonlinear modeling and focused on the combination of filters and nonlinearities. Several applications demonstrate the importance of nonlinear processors. The basic building blocks of the previous chapters such as filters, delays and modulators/demodulators are particularly useful for exploring new improved nonlinear processors.

Sound and Music

- [m-Bec89] Jeff Beck: *Guitar Shop*. 1989.
- [m-Bla70] Deep Purple: *Deep Purple in Rock*. 1970.
- [m-Cla67] Cream: *Disraeli Gear*. 1967.
- [m-Hen67a] Jimi Hendrix: *Are You Experienced?* 1967.
- [m-Hen67b] Jimi Hendrix: *Bold As Love*. 1967.
- [m-Hen68] Jimi Hendrix: *Electric Ladyland*. 1968.
- [m-Pag69] Led Zeppelin: *I/II/III/IV*. 1969-1971.
- [m-San99] Santana: *Supernatural*. 1999.

Bibliography

- [Alt90] S.R. Alten. *Audio in Media*. Wadsworth, 1990.
- [And95] C. Anderton. *Multieffects for Musicians*. Amsco Publications, 1995.
- [Arf79] D. Arfib. Digital synthesis of complex spectra by means of multiplication of nonlinear distorted sine waves. *J. Audio Eng. Soc.*, 27:757-768, October 1979.
- [Bar98] E. Barbour. The cool sound of tubes. *IEEE Spectrum*, pp. 24-32, August 1998.
- [Ben97] R. Bendiksen. Digitale Lydeffekter. Master's thesis, Norwegian University of Science and Technology, 1997.

- [Bru79] M. Le Brun. Digital waveshaping synthesis. *J. Audio Eng. Soc.*, 27:250–265, April 1979.
- [De 84] G. De Poli. Sound synthesis by fractional waveshaping. *J. Audio Eng. Soc.*, 32(11):849–861, November 1984.
- [Dic87] M. Dickreiter. *Handbuch der Tonstudiatechnik, Band I und II*. K.G. Saur, 1987.
- [Doy93] M. Doyle. *The History of Marshall*. Hal Leonard Publishing Corporation, 1993.
- [Ear76] J. Eargle. *Sound Recording*. Van Nostrand, 1976.
- [Fli93] R. Fliegler. *AMPS! The Other Half of Rock'n'Roll*. Hal Leonard Publishing Corporation, 1993.
- [Fra97] W. Frank. *Aufwandsarme Modellierung und Kompensation nichtlinearer Systeme auf Basis von Volterra-Reihen*. PhD thesis, University of the Federal Armed Forces Munich, Germany, 1997.
- [FUB⁺98] A. Farino, E. Ugolotti, A. Bellini, G. Cibelli, and C. Morandi. Inverse numerical filters for linearisation of loudspeaker responses. In *Proc. DAFX-98 Digital Audio Effects Workshop*, pp. 12–16, Barcelona, November 1998.
- [Ham73] R.O. Hamm. Tubes versus transistors — is there an audible difference? *J. Audio Eng. Soc.*, 21(4):267–273, May 1973.
- [Kai87] A.J.M. Kaizer. Modelling of the nonlinear response of an electrodynamic loudspeaker by a volterra series expansion. *J. Audio Eng. Soc.*, 35(6):421–433, 1987.
- [Kli98] W. Klippel. Direct feedback linearization of nonlinear loudspeaker systems. *J. Audio Eng. Soc.*, 46(6):499–507, 1998.
- [McN84] G.W. McNally. Dynamic range control of digital audio signals. *J. Audio Eng. Soc.*, 32(5):316–327, May 1984.
- [Nie00] S. Nielsen. Personal communication. TC Electronic A/S, 2000.
- [Orf96] S.J. Orfanidis. *Introduction to Signal Processing*. Prentice-Hall, 1996.
- [PD93] D. Peterson and D. Denney. *The VOX Story*. The Bold Strummer Ltd., 1993.
- [Rat95] L. Ratheiser. *Das große Röhrenhandbuch*. Reprint. Franzis-Verlag, 1995.
- [RCA59] RCA. *Receiving Tube Manual. Technical Series RC-19*. Radio Corporation of America, 1959.

- [RH96] M.J. Reed and M.O. Hawksford. Practical modeling of nonlinear audio systems using the volterra series. In *Proc. 100th AES Convention*, Preprint 4264, 1996.
- [RZ95] B. Redmer and U. Zölzer. *Parameters for a Dynamic Range Controller*. Technical report, Hamburg University of Technology, 1995.
- [Sch80] M. Schetzen. *The Volterra and Wiener Theories of Nonlinear Systems*. Robert Krieger Publishing, 1980.
- [SZ99] J. Schattschneider and U. Zölzer. Discrete-time models for nonlinear audio systems. In *Proc. DAFX-99 Digital Audio Effects Workshop*, pp. 45–48, Trondheim, December 1999.
- [vdL97] R. von der Linde. *Röhrenverstärker*. Elektor-Verlag, 1997.
- [WG94] M. Warstat and T. Görne. *Studiotechnik — Hintergrund und Praxiswissen*. Elektor-Verlag, 1994.
- [Whi93] P. White. *L'enregistrement créatif, Effets et processeurs, Tomes 1 et 2*. Les cahiers de l'ACME, 1993.
- [ZF90] E. Zwicker and H. Fastl. *Psychoacoustics*. Springer-Verlag, 1990.
- [Zöl97] U. Zölzer. *Digital Audio Signal Processing*. John Wiley & Sons, Ltd, 1997.

Chapter 6

Spatial Effects

D. Rocchesso

6.1 Introduction

The human peripheral hearing system modifies the sound material that is transmitted to the higher levels in the brain, and these modifications are dependent on the incoming direction of the acoustic waves. From the modified sound signals several features are collected in a set of spatial cues, used by the brain to infer the most likely position of the sound source. Understanding the cues used by the hearing system helps the audio engineer to introduce some artificial features in the sound material in order to project the sound events in space. In the first half of this chapter, the most important techniques for sound projection are described, both for individual listeners using headphones and for an audience listening through a set of loudspeakers.

In natural listening conditions, sounds propagate from a source to a listener and during this trip they are widely modified by the environment. Therefore, there are some spatial effects imposed by the physical and geometric characteristics of the environment to the sound signals arriving at the listener's ears. Generally speaking, we refer to the kind of processing operated by the environment as reverberation. The second half of this chapter illustrates these kinds of effects and describes audio processing techniques that have been devised to imitate and extend the reverberation that occurs in nature.

The last section of the chapter describes a third category of spatial effects that is inspired by the natural spatial processing of sounds. In this miscellaneous category we list all the filtering effects that are designed to change the apparent source width, the spaciousness of a sound field, and the directivity of a loudspeaker set. Moreover, the geometric and physical characteristics of reverberating enclosures are used as design features for generalized resonators to be used as part of sound synthesis algorithms.

The importance of space has been largely emphasized in electro-acoustic compositions, but sophisticated spatial orchestrations often result in poor musical messages to the listener. Indeed, space cannot be treated as a composition parameter in the same way as pitch or timbre are orchestrated, just because space for sounds is not an “indispensable attribute” [KV01] as it is for images. This relative weakness is well explained if we think of two loudspeakers playing the same identical sound track: the listener will perceive one apparent source. The phenomenon is analogous to two colored spotlights that fuse to give one new, apparent, colored spot. In fact, color is considered a non-indispensable attribute for visual perception. However, just as color is a very important component in visual arts, the correct use of space can play a fundamental role in music composition, especially in improving the effectiveness of other musical parameters of sound, such as pitch, timbre, and intensity.

6.2 Basic Effects

6.2.1 Panorama

Introduction

Using a multichannel sound reproduction system we can change the apparent position of a virtual sound source just by feeding the channels with the same signal and adjusting the relative amplitude of the channels. This task is usually accomplished, as part of the mixdown process, by the sound engineer for each sound source, thus composing a *panorama* of acoustic events in the space spanned by the loudspeakers.

Acoustic and Perceptual Foundations

For reproduction via multiple loudspeakers, it is important to take some specific aspects of localization into account. In fact, different and sometimes contradictory cues come into play when multiple sources radiate coherent or partially coherent signals [Bla83]. Particularly important is the case of a listener hearing signals arriving from the sources only at slightly different levels and times. In this case the sources will cooperate to provide a single sound event located at a place different from the source locations. For larger differences in the incoming signal the virtual sound images tend to collapse onto one of the real sources. The precedence effect (see section 6.2.2) is largely responsible for this phenomenon.

Figure 6.2 shows the kind of curves that it is possible to draw from experiments with a standard stereo layout (i.e. central listener and angle of 60° with the loudspeakers, as in Fig. 6.1) and with broadband impulses as test signals [Bla83]. Figure 6.2.a shows the perceived displacement for a given level difference. Figure 6.2.b shows the perceived displacement for a given time difference. The curve for level difference is well approximated by the Blumlein law [Bla83]

$$\sin \theta = \frac{g_L - g_R}{g_L + g_R} \sin \theta_l , \quad (6.1)$$

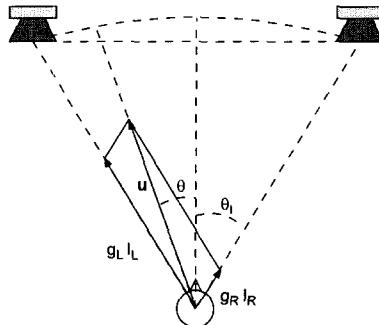


Figure 6.1 Stereo panning. θ is the angle of the apparent source position.

or by the tangent law

$$\tan \theta = \frac{g_L - g_R}{g_L + g_R} \tan \theta_l , \quad (6.2)$$

where g_L and g_R are the gains to be applied to the left and right stereo channels, θ is the angle of the virtual source position, and θ_l is the angle formed by each loudspeaker with the frontal direction. In [Pul97] it is shown that the tangent law results from a vector formulation of amplitude panning (see section 6.4.3). This formulation, as well as the curves of Fig. 6.2 are mainly valid for broadband signals or at low frequencies (below 500-600 Hz). For narrowband signals at higher frequencies the curves are quite different and the curve for time differences can even be nonmonotonic [Bla83].

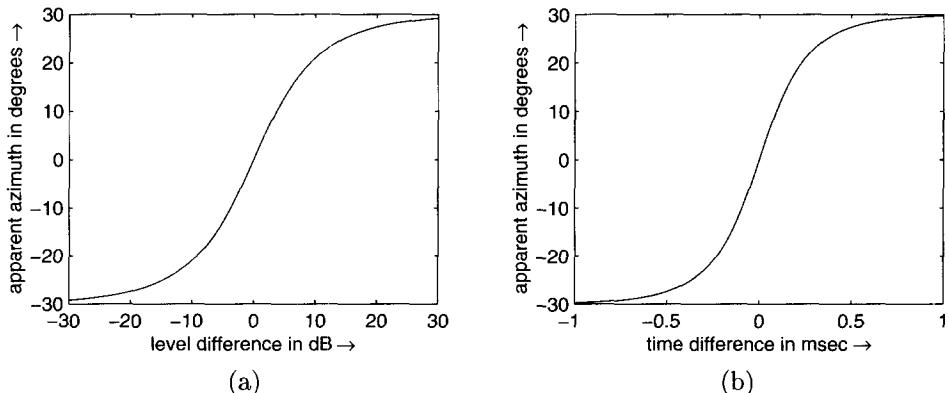


Figure 6.2 Perceived azimuth of a virtual sound source when a standard stereo layout is driven with signals that only differ in level (a) or in time delay (b).

Signal Processing

In a standard stereo loudspeaker set-up it is assumed that the listener stands in central position and forms an angle $2\theta_l$ with the two loudspeakers (see Fig. 6.1). Two gains g_L and g_R should be applied to the left and right channel, respectively, in order to set the apparent azimuth at the desired value θ . A unit-magnitude two-channel signal, corresponding to the central apparent source position ($\theta = 0$), can be represented by the column vector

$$\mathbf{u} = \begin{bmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix}, \quad (6.3)$$

so that the gains to be applied to the two channels in order to steer the sound source to the desired azimuth are obtained by the matrix-vector multiplication:

$$\begin{bmatrix} g_L \\ g_R \end{bmatrix} = \mathbf{A}_\theta \mathbf{u}. \quad (6.4)$$

The matrix \mathbf{A}_θ is a rotation matrix. If $\theta_l = 45^\circ$ the rotation matrix takes the form

$$\mathbf{A}_\theta = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}, \quad (6.5)$$

so that when $\theta = \pm\theta_l$ only one of the two channels is non zero. It is easily verified that the rotation by matrix (6.5) corresponds to applying the tangent law (6.2) to the configuration with $\theta_l = 45^\circ$. Amplitude panning by means of a rotation matrix preserves the loudness of the virtual sound source while moving its apparent azimuth. In contrast, linear cross-fading between the two channels does not preserve the loudness and determines a “hole in the middle” of the stereo front.

The matrix-based panning of equation (6.4) is suited to direct implementation in Matlab. The M-file 6.1 implements the amplitude panning between an initial and a final angle.

```
M-file 6.1 (matpan.m)
initial_angle = -40; %in degrees
final_angle = 40;    %in degrees
segments = 32;
angle_increment = (initial_angle - final_angle)/segments * pi / 180;
                    % in radians
lenseg = floor(length(monosound)/segments) - 1;
pointer = 1;
angle = initial_angle * pi / 180; %in radians

for i=1:segments
    A =[cos(angle), sin(angle); -sin(angle), cos(angle)];
    x = [monosound(pointer:pointer+lenseg);
          monosound(pointer:pointer+lenseg)];
```

```

y = [y, A * x];
angle = angle + angle_increment; pointer = pointer + lenseg;
end;

```

The monophonic input sound (stored in array `monosound`) is segmented in a number `segments` of blocks and each block is rotated by means of a matrix-by-vector multiplication.

In practice, the steering angle θ does not necessarily correspond to the perceived localization azimuth. The perceived location is influenced by the frequency content of sound. Some theories of “directional psychoacoustics” have been developed in the past in order to drive the rotation matrices with the appropriate coefficients [Ger92a]. Accurate implementations use frequency-dependent rotation matrices, at least discriminating between low (less than about 500 Hz) and high (between 500 Hz and 3500 Hz) frequency components [PBJ98]. We will discuss directional psychoacoustics in further detail in section 6.4.4, in the context of Ambisonics surround sound.

Music Applications and Control

Stereo panning has been applied to recordings of almost every music genre. In some cases, the panoramic knob of domestic hi-fi systems has been exploited as an artistic resource. Namely, the composition HPSCHD by John Cage and Lejaren Hiller appears in the record [m-Cag69] accompanied by a sheet (unique for each record copy) containing a computer-generated score for the listener. In the score, panoramic settings are listed at intervals of 5 seconds together with values for the treble and bass knobs.

6.2.2 Precedence Effect

Introduction

In a stereo loudspeaker set-up, if we step to one side of the central position and listen to a monophonic music program, we locate the apparent sound source in the same position as our closest loudspeaker, and the apparent position does not move even if the other channel is significantly louder. The fact that the apparent source collapses into one of the loudspeakers is due to the precedence effect, a well-known perceptual phenomenon that can be exploited in certain musical situations.

Acoustic and Perceptual Foundations

Our hearing system is very sensitive to the direction of the first incoming wavefront, so that conflicting cues coming after the first acoustic impact with the source signal are likely to be ignored. It seems that spatial processing is inhibited for a few tens of milliseconds after the occurrence of a well-localized acoustic event [Gri97]. This is the precedence effect [Bla83]. Clearly, the first wavefront is related to a transient

in sound production, and transients are wideband signals exciting a wide frequency range where the directional properties of the outer ear (see section 6.3.4) can be used to give a hint of incoming direction.

Signal Processing

In a standard stereo loudspeaker set-up, with the listener in central position as in Fig. 6.1, a monophonic source material that feeds both loudspeakers with the same loudness can be moved towards one of the loudspeakers just by inserting some delay in the other channel. Figure 6.2.b shows the qualitative dependency of the apparent azimuth on the relative delay between channels. The actual curve strongly depends on the kind of sounds that are played [Bla83].

A simple circuit allowing a joint control of panorama and precedence effect is shown in Fig. 6.3. The variable-length delay lines should allow delays up to 1ms.

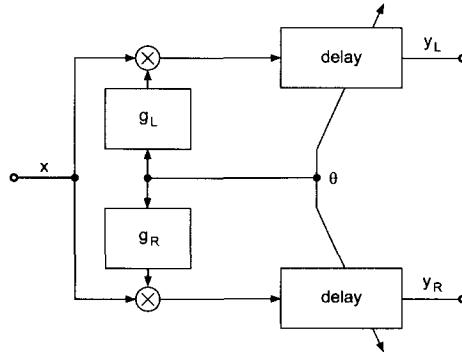


Figure 6.3 Control of panorama and precedence effect.

Music Applications and Control

In a live electro-acoustic music performance, it is often desirable to keep all the apparent sound sources in the front, so that they merge nicely, both acoustically and visually, with the sounds coming from acoustic instruments on stage. However, if only the front loudspeakers are playing it might be difficult to provide all the audience with a well-balanced sound, even at the seats in the rear. A good solution, called *transparent amplification* by Alvise Vidolin [RV96], is to feed the rear loudspeaker with a delayed copy of the front signals. The precedence effect will ensure that the apparent location is on stage as long as the delay of the rear signals is at least as long as the time taken by sounds to go from the stage to the rear loudspeakers. Given the length of the room L and the speed of sound $c \approx 340\text{m/s}$, the delay in seconds should be about

$$T_D = \frac{L}{c}. \quad (6.6)$$

6.2.3 Distance and Space Rendering

Introduction

In digital audio effects, the control of apparent distance can be effectively introduced even in monophonic audio systems. In fact, the impression of distance of a sound source is largely controllable by insertion of artificial wall reflections or reverberant room responses.

Acoustic and Perceptual Foundations

There are no reliable cues for distance in anechoic or open space. Familiarity with the sound source can provide distance cues related with air absorption of high frequency. For instance, familiarity with a musical instrument tells us what the average intensity of its sounds are when coming from a certain distance. The fact that timbral qualities of the instrument will change when playing loudly or softly is also a cue that helps with the identification of distance. These cues seem to vanish when using unfamiliar sources or synthetic stimuli that do not resemble any physical sounding object. Conversely, in an enclosure the ratio of reverberant to direct acoustic energy has proven to be a robust distance cue [Bla83]. It is often assumed that in a small space the amplitude of the reverberant signal changes little with distance, and that in a large space it is roughly proportional to $1/\sqrt{\text{Distance}}$ [Cho71]. The direct sound attenuates as $1/\text{Distance}$ if spherical waves are propagated.

Special, frequency-dependent cues are used when the source is very close to the head. These cues are briefly described in section 6.3.2.

Signal Processing

A single reflection from a wall can be enough to provide some distance cues in many cases. The physical situation is illustrated in Fig. 6.4a, together with a signal processing circuit that reproduces it. A single delay line with two taps is enough to reproduce this basic effect. Moreover, if the virtual sound source is close enough to the listening point, the first tap can be taken directly from the source, thus

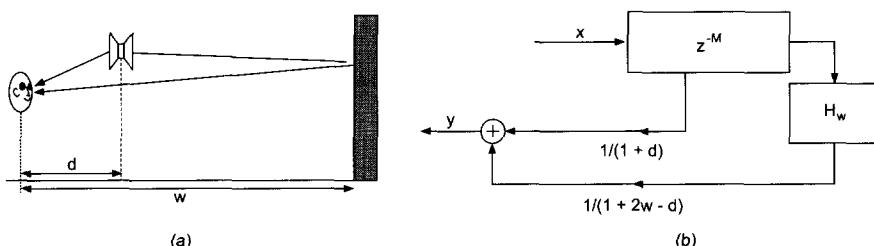


Figure 6.4 Distance rendering via single wall reflection: (a) physical situation, (b) signal processing scheme.

reducing the signal processing circuitry to a simple non-recursive comb filter. To be physically consistent, the direct sound and its reflection should be attenuated as much as the distance they travel, and the wall reflection should also introduce some additional attenuation and filtering in the reflected sound, represented by the filter H_w in Fig. 6.4b. The distance attenuation coefficients of Fig. 6.4b have been set in such a way that they become one when the distance goes to zero, just to avoid the divergence to infinity that would come from the physical laws of a point source.

From this simple situation it is easy to see how the direct sound attenuates faster than the reflected sound as long as the source approaches the wall.¹ This idea can be generalized to closed environments adding a full reverberant tail to the direct sound. An artificial yet realistic reverberant tail can be obtained just by taking an exponentially decayed gaussian noise and convolving it with the direct sound. The reverberant tail should be added to the direct sound after some delay (proportional to the size of the room) and should be attenuated with distance in a lesser extent than the direct sound. Figure 6.5 shows the signal processing scheme for distance rendering via room reverberation.

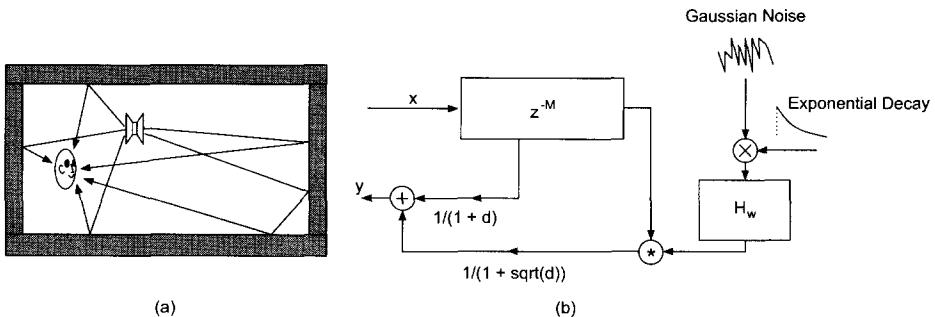


Figure 6.5 Distance rendering via room reverberation: (a) physical situation, (b) signal processing scheme.

The following M-file 6.2 allows experimentation with the situations depicted in Fig. 6.4 and 6.5, with different listener positions, provided that x is initialized with the input sound, and y , z , and w are long-enough vectors initialized to zero.

M-file 6.2 (distspace.m)

```

h = filter([0.5,0.5],1, ...
    random('norm',0,1,1,lenh).*exp(-[1:lenh]*0.01/distwall)/100);
    % reverb impulse response
offset = 100;
st = Fs/2;

for i = 1:1:distwall-1 % several distances listener-source
    del1 = floor(i/c*Fs);

```

¹Indeed, in this single reflection situation, the intensity of the reflected sound increases as the source approaches the wall.

```

del2 = floor((distwall*2 - i)/c*Fs);
y(i*st+1:i*st+del1) = zeros(1,del1);
y(i*st+del1+1:i*st+del1+length(x)) = x./(1+i); % direct signal
w(i*st+del2+1:i*st+del2+length(x)) = ...
    y(i*st+del2+1:i*st+del2+length(x)) + ...
    x./(1+(2*distwall-i)); % direct signal + echo
z(i*st+del2+1:i*st+del2+length(x)+lenth-1+offset) = ...
    y(i*st+del2+1:i*st+del2+length(x)+lenth-1+offset) + ...
    [zeros(1,offset),conv(x,h)]./sqrt(1+i);
                                % direct signal + delayed reverb
end

```

Music Applications and Control

An outstanding example of musical control of distance is found in the piece “Turenas” by John Chowning [m-Cho72], where the reverberant sound amplitude is decreased with the reciprocal of the square root of distance, much slower than the decrease of direct sound. In the composition, distance cues are orchestrated with direction (as in section 6.2.1) and movement (as in section 6.2.4) cues.

Panorama, precedence effect, and distance rendering taken as a whole contribute to the definition of a sonic perspective [Cho99]. The control of such sonic perspective helps the task of segregation of individual sound sources and the integration of music and architecture. A noteworthy example of fusion between architecture and music is found in the piece “Prometeo”, by Luigi Nono [m-Non82]. In the original performance the players were located in different positions of an acoustic shell designed by architect Renzo Piano, and sounds from live-electronic processing were subject to spatial dislocation in a congruent sonic architecture.

Another important piece that makes extensive use of space rendering is “Répons”, by Pierre Boulez [m-Bou84], which is performed with an unusual arrangement of soloists, choir, and audience. The first spatial element exploited in the composition is the physical distance between each soloist and the choir. Moreover, the sounds of the soloists are analyzed, transformed, and spatialized by means of a computer and a set of six loudspeakers. For the CD, sounds have been processed using the software “Spatialisateur” [Jot99] in order to place the orchestra in front of the listener and to enlarge the audio image of the soloists. The Spatialisateur, implemented at IRCAM in Paris, makes use of panorama and distance rendering via room reverberation, using sophisticated techniques such as those described in section 6.5.

6.2.4 Doppler Effect

Introduction

Movements of the sound sources are detected as changes in direction and distance cues. Doppler effect is a further (strong) cue that intervenes whenever there is a radial component of motion between the sound source and the listener. In a closed

environment, radial components of motion are likely to show up via reflections from the walls. Namely, even if a sound source is moving at constant distance from the listener, the paths taken by the sound waves via wall reflections are likely to change in length. If the source motion is sufficiently fast, in all of these cases we have transpositions in frequency of the source sound.

Acoustic and Perceptual Foundations

The principle of the Doppler effect is illustrated in Fig. 6.6, where the listener is moving toward the sound source with speed c_s . If the listener meets f_s wave crests per second at rest, he ends up meeting crests at the higher rate

$$f_d = f_s \left(1 + \frac{c_s}{c}\right) \quad (6.7)$$

when the source is moving. Here c is the speed of sound in air. We usually appreciate the pitch shift due to Doppler effect in non-musical situations, such as when an ambulance or a train is passing by. The perceived cue is so strong that it can evoke the relative motion between source and listener even when other cues indicate a relative constant distance between the two. In fact, ambulance or insect sounds having a strong Doppler effect are often used to demonstrate how good a spatialization system is, thus deceiving the listener who does not think that much of the spatial effect is already present in the monophonic recording. Recent research in psychoacoustics has also shown how the perception of pitch can be strongly affected by dynamic changes in intensity, as are found in situations where the Doppler effect occurs [Neu98]. Namely, a sound source approaching the listener at constant speed determines a rapid increase in intensity when it traverses the neighborhood of the listener. On the other hand, while the frequency shift is constant and positive before passing the listener, and constant and negative after it has passed, most listeners perceive an increase in pitch shift as the source is approaching. Such apparent pitch increase is due to the simultaneous increase in loudness.

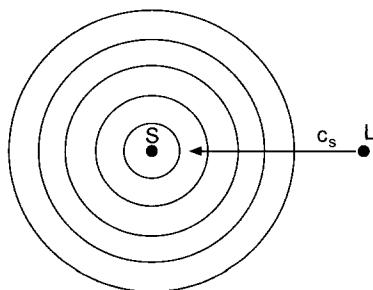


Figure 6.6 Illustration of the Doppler effect.

Signal Processing

The Doppler effect can be faithfully reproduced by a pitch shifter (see Chapter 7) controlled by the relative velocity between source and listener. In particular, the circuit of Fig. 7.16 can be used with sawtooth control signals whose slope increases with the relative speed. Figure 6.7 shows the signal to be used to control one of the delays of Fig. 7.16 for a sound source that approaches the listening point and passes it. Before the source reaches the listener, the sound is raised in pitch, and it is lowered right after.

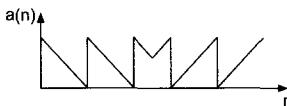


Figure 6.7 Control signal for simulating the Doppler effect with a delay-based pitch shifter.

Any sound processing model based on the simulation of wave propagation, such as the model described in section 6.4.6, implements an implicit simulation of the Doppler effect. In fact, these models are based on delay lines that change their length according to the relative position of source and listener, thus providing positive or negative pitch transpositions.

In general, the accuracy and naturalness of a Doppler shift reproduced by digital means depend on the accuracy of interpolation in variable-length delays. If this is not good enough, modulation products affect the transposed signal producing remarkable artifacts. The enhanced techniques described in sections 3.2.4 and 7.4 can be applied for various degrees of fidelity.

Music Application and Control

The motion of source or listener in musical contexts has to be performed with care, since uncontrolled frequency shifts completely destroy any harmonic relations between the notes of a melody. However, there are notable examples of electro-acoustic music works incorporating Doppler effects [m-Cho72, m-Pis95]. In most cases, the sounds that are subjected to Doppler shifting are various kinds of noises or percussive patterns, so that they do not impose any special harmonic gliding character on the piece.

6.2.5 Sound Trajectories

The first attempt to design spatial movements as an independent dimension of music composition is probably Stockhausen's "Gesang der Jünglinge" [m-Sto56], where directions and movements among five groups of loudspeakers, arranged in a circle around the audience, were precisely programmed by the author. This piece is also probably the origin of the "spatial utopia" of a large part of contemporary music,

since Stockhausen gave abundant and clear explanations of his aesthetics, where velocities and spatial displacements are considered to be as important as pitches and note durations. This suggestive idea seems to ignore some perceptual evidence and the difficulties related to conveying consistent spatial information to an audience. Nevertheless, sound trajectories in [m-Sto56] are certainly essential to the dramatic development of the piece.

Since the work of Chowning [Cho71, m-Cho72], many computer systems have been built in order to help the musician to compose with space. Most of these systems are based on a software interface that allows the musician to display or/and define trajectories of virtual sound sources in the listening space, where actual loudspeakers are positioned. In fact, the manipulation of sound trajectories belongs to a control layer built on top of the signal processing level that implements panoramic or distance effects. Parameters such as angular position, distance from loudspeakers, etc., are taken directly from the curves at a rate that is high enough to ensure smooth movements without noticeable artifacts. In this respect, if only panoramic and distance effects are affected by trajectories, the control rate can be very low (e.g., 20 Hz) thus allowing separation of the control layer from the signal processing layer via a low-bandwidth communication channel. Vice versa, if the Doppler effect is somehow taken into account, the control rate must be much higher, because the ear is sensitive to minuscule variations of sound pitch. Alternatively, control signals can still be transmitted at low rate if at the signal processing end there is an interpolation mechanism that reconstructs the intermediate values of control parameters.

Sometimes spatial trajectories are taken as a metaphor for guiding transformations of sonic materials. For instance, Truax [m-Tru85] uses multiple epicycles to control transformations of continuous sounds and to project them into space. In this example, the consistency between spatial display and sound transformation helps the listener to discriminate and follow several music lines without the aid of pitch or rhythmic patterns.

Software systems for the definition and display of sound trajectories are commonly used in live electronics. For instance, Belladonna and Vidolin developed an interface for two-dimensional panoramic control for the visual language MAX [BV95]. Trajectories can be memorized, recalled, and translated at runtime into MIDI messages sent to a MIDI-controlled audio mixer. External reverberating devices can also be controlled and synchronized with panoramic messages in order to recreate an impression of depth and distance. This system has been used in several music productions and in various listening spaces. Figure 6.8 illustrates two erratic movements of virtual sound sources (the actual sources are the two percussion sets) in the big San Marco church in Milan, as it was programmed for a composition by Guarnieri [m-Gua99]. This piece and listening space are noteworthy for the use of intensity-controlled sound panning over a wide area, i.e., some sources move from front to back as they get louder. In general, sound trajectories are rarely meaningful *per se*, but they can become a crucial dramatic component if their variations are controlled dynamically as a function of other musical parameters such as intensity, or rhythm.

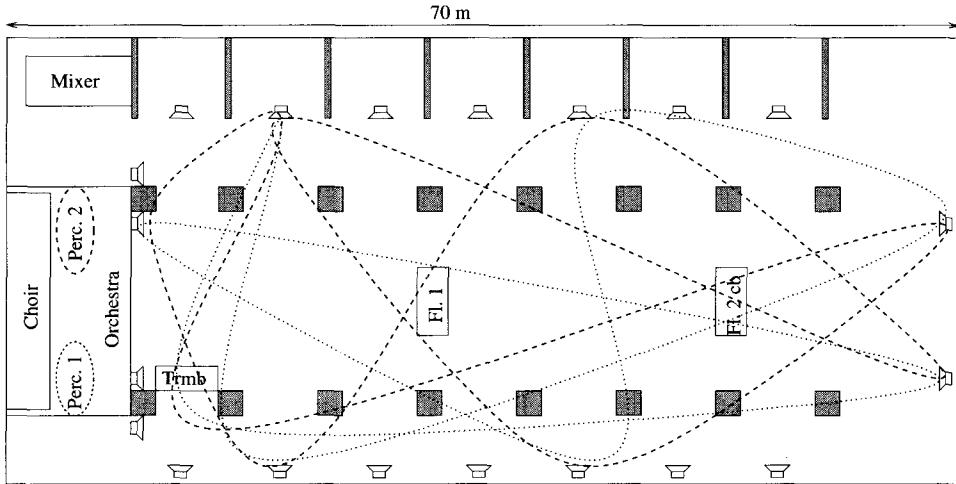


Figure 6.8 Spatial layout and sonic trajectories for the performance of a composition by G. Guarnieri.

A complex system to control sound trajectories in real-time was implemented on the IRIS-MARS workstation [RBM95]. It implemented the room-within-the-room model described in section 6.4.6 by reserving one of the two processors for spatialization, and the other for reverberation. The coordinates of source motion and other parameters of spatialization could be precomputed or controlled in real-time via MIDI devices. This kind of spatial processing has been used in the piece “Games” by F. Cifariello Ciardi [m-Cif95].

A sophisticated system providing real-time multichannel spatial audio processing on a computer workstation is the “Spatialisateur” developed at IRCAM in Paris [Jot92, Jot99]. In this system, an intermediate level is interposed between the signal processing level and the control level. Namely, the physical parameters of virtual rooms are accessed through a set of perceptual attributes such as warmth, brilliance, etc., in such a way that compositional control of space becomes easier. As the interface is provided in the visual language MAX, it is easy to associate sets of parameters to trajectories that are drawn in a 2-D space.

6.3 3D with Headphones

6.3.1 Localization

Introduction

Humans can localize sound sources in a 3D space with good accuracy using several cues. If we can rely on the assumption that the listener receives the sound material via a stereo headphone, we can reproduce most of the cues that are due to the filter-

ing effect of the pinna-head-torso system, and inject the signal artificially affected by this filtering process directly to the ears.

Acoustic and Perceptual Foundations

Classic psychoacoustic experiments have shown that, when excited with simple sine waves, the hearing system uses two strong cues to estimate the apparent direction of a sound source. Namely, interaural intensity and time differences (IID and ITD) are jointly used to that purpose. IID is mainly useful above 1500 Hz, where the acoustic shadow produced by the head becomes effective, thus reducing the intensity of the waves reaching the contralateral ear. For this high-frequency range and for stationary waves, the ITD is also far less reliable, since it produces phase differences in sine waves which often exceed 360° . Below 1500 Hz the IID becomes smaller due to head diffraction which overcomes the shadowing effect. In this low-frequency range it is possible to rely on phase differences produced by the ITD. IID and ITD can only partially explain the ability to discriminate among different spatial directions. In fact, if the sound source moved laterally along a circle (see Fig. (6.9)) the IID and ITD would not change. The cone formed by the circle with the center of the head has been called *cone of confusion*. Front-back and vertical discrimination within a cone of confusion are better understood in terms of broadband signals and Head-Related Transfer Functions (HRTF). The system pinna-head-torso acts like a linear filter for a plane wave coming from a given direction. The magnitude and phase responses of this filter are very complex and direction dependent, so that it is possible for the listener to disambiguate between directions having the same, stationary, ITD and IID. In some cases, it is advantageous to think about these filtering effects in the time domain, thus considering the Head-Related Impulse Responses (HRIR).

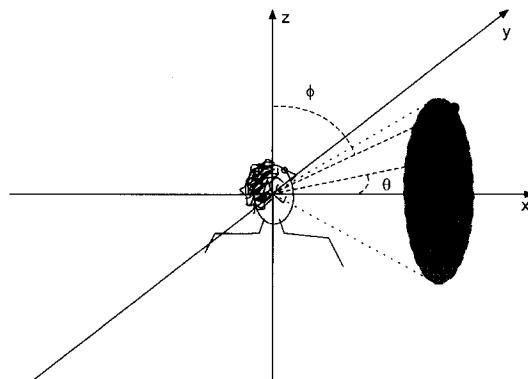


Figure 6.9 Interaural polar coordinate system and cone of confusion.

In section 6.3.4, we describe a structural model for HRTFs [BD98] where it is possible to outline a few relevant direction-dependent cues obtained from simplification of HRTFs. Even if the cues are simplified, when they are varied dynamically they can give a strong impression of localization. In real life, the listener is never

static when listening to a sound source. Even small movements of the head greatly help in discriminating possible confusions, such as the uncertainty between a central source in front of the listener or the same source exactly behind the listener. Therefore, a small set of cues such as ITD, IID, and the major notches of HRTFs can be sufficient to give a strong impression of direction, as long as the cues are related to movements of the listener's head.

6.3.2 Interaural Differences

Sound spatialization for headphones can be based on interaural intensity and time differences. The amplitude and time delay of each channel should just be governed by curves similar to those of Fig. 6.2. It is possible to use only one of the two cues, but using both cues will provide a stronger spatial impression. Of course, interaural time and intensity differences are just capable of moving the apparent azimuth of a sound source, without any sense of elevation. Moreover, the apparent source position is likely to be located inside the head of the listener, without any sense of externalization. Special measures have to be taken in order to push the virtual sound sources out of the head.

A finer localization can be achieved by introducing frequency-dependent interaural differences. In fact, due to diffraction the low frequency components are barely affected by IID, and the ITD is larger in the low frequency range. Calculations done with a spherical head model and a binaural model [Kuh77, PKH99] allow the drawing of approximated frequency-dependent ITD curves, one being displayed in Fig. 6.10a for 30° of azimuth. The curve can be further approximated by constant segments, one corresponding to a delay of about 0.38 ms at low frequencies, and the other corresponding to a delay of about 0.26 ms at high frequencies. The low-frequency limit can in general be obtained for a general incident angle θ by the formula

$$\text{ITD} = \frac{1.5\delta}{c} \sin \theta , \quad (6.8)$$

where δ is the inter-ear distance in meters and c is the speed of sound. The crossover point between high and low frequencies is located around 1 kHz. Similarly, the IID should be made frequency dependent. Namely, the difference is larger for high-frequency components, so that we have IID curves such as that reported in Fig. 6.10b for 30° of azimuth. The IID and ITD are shown to change when the source is very close to the head [DM98]. In particular, sources closer than five times the head radius increase the intensity difference in low frequency. The ITD also increases for very close sources but its changes do not provide significant information about source range.

6.3.3 Externalization

Listening to binaural sound material through loudspeakers often causes internalization of the sound sources in the head of the listener. It seems that human subjects

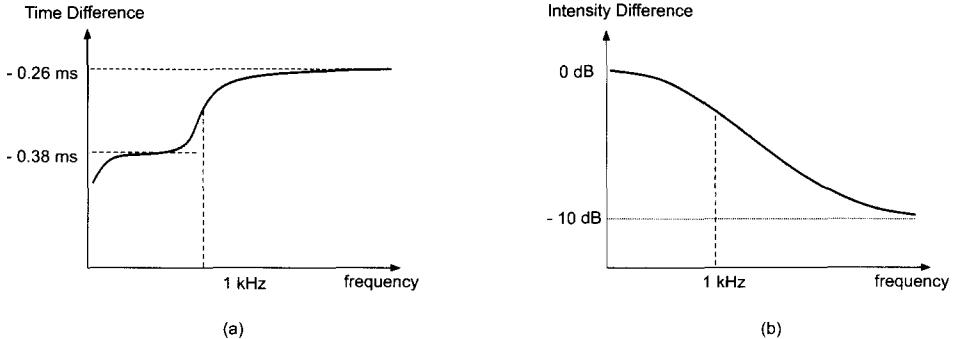


Figure 6.10 Frequency-dependent interaural time (a) and intensity (b) difference for azimuth 30° .

tend to internalize the perceived objects when the total stimulation, as coming from all sensorial modes, cannot be produced by natural situations involving distant sources [Dur92]. One technique that is recognized to be effective in externalizing the sound sources when headphones are used, is decorrelation [Ken95b], and it is justified by the fact that, in natural situations, the signals reaching the ears are significantly decorrelated, especially because of room reverberation. The correlation of the left and right channels is represented by the function

$$r(\tau) = \lim_{T \rightarrow \infty} \frac{1}{2T\nu} \int_{-T}^{+T} x_L(t)x_R(t + \tau)dt . \quad (6.9)$$

The argument τ is the time lag between the two channels and ν is a normalizing factor defined as

$$\nu = \frac{1}{2T} \sqrt{\int_{-T}^{+T} x_L^2(t)dt \int_{-T}^{+T} x_R^2(t)dt} . \quad (6.10)$$

If one channel is just a shifted copy of the other, the correlation function will have a strong peak for a value of the argument equal to the time shift. Usually the degree of correlation is measured by a single number taken as the absolute value of the maximum of the correlation function. The normalization factor is chosen in order to produce a degree of correlation equal to 1 for pure shift and -1 for signals that are 180° out of phase. Two signals having a degree of correlation equal to 1 in magnitude are called coherent. The coherence between the left and right channels is maintained under time and intensity differences, and even under phase inversion.

When a subject listens to two channels of broadband noise via headphones, the relative degree of correlation produces a spatial image that ranges from central inside the head to lateral right out of the ears (see Fig. 6.11). Partially coherent signals produce images that are larger and less sharply located [Bla83] than those of perfectly coherent signals. The degree of correlation is generally reduced in presence of reverberation. Thus, the techniques that are commonly used for artificial reverberation can also be used to decorrelate two channels. Of course, if the only purpose

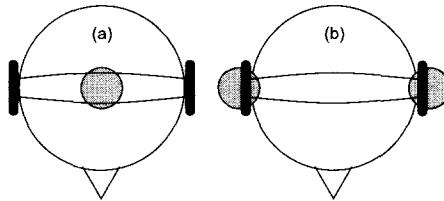


Figure 6.11 Perceived sound image when listening to broadband noise with degree of correlation equal to 1 (a), and 0 (b).

is to externalize a sound image, we want to avoid other artifacts such as excessive coloration or reverberant tail. We will see in section 6.6.1 how a good decorrelator can be built. It can be applied both to externalize a sound source in binaural listening and to adjust the apparent source width in listening via loudspeakers.

The degree of correlation, beside helping to push the apparent sources out of the head, is also useful in arranging an ensemble of sources in space. In fact, it has been shown by experiments that the listener assigns a unique spatial image to sound sources having similar degree of correlation. For instance, a coherent couple of voice signals added to the broadband noise of Fig. 6.11b, produces a distinct image in the center of the head. This property of the hearing system can be exploited to augment the clarity of display of a mixture of sound sources.

A dramatic increase in externalization is achieved if the binaural stimulus is dynamically varied to follow the head movements in a consistent, natural way [Dur92]. However, this control of spatialization based on head tracking is too expensive and too cumbersome in most cases. Therefore, it is usually assumed that the listener keeps her head fixed while listening, for instance because she is watching a screen in a standard computer desktop environment [Kyr98]. In this situation, the simultaneous presentation of visual stimuli correlated with sounds is also known to help externalize the auditory events.

Recently, excellent results in localization accuracy and externalization have been reported in anechoic rendering via headphones under fixed head conditions, even when the details of the filtering characteristics of the head are smoothed significantly [KC98]. It seems that if the playback apparatus preserves the acoustics of a natural listening condition the sound sources do externalize in most cases. Unfortunately, such a playback apparatus requires the correct coupling with the ear-canal impedance and this is not what happens with conventional headphones. Therefore, the control over the degree of correlation, even if its effectiveness and accuracy are limited, seems to be the only way we have to externalize a sound source for a generic listener. If we can afford signal processing specialized to the listener's head and headphone characteristics, better accuracy can be achieved by modeling the HRTF. The effects of binaural spectral details on externalization were extensively investigated in [HW96], where it was shown that sources are externalized well even with frequency-independent ITD while, on the other hand, the effects of IID are accumulated over the entire spectrum.

6.3.4 Head-Related Transfer Functions

Several authors have measured the filtering properties of the system pinna-head-torso by means of manikins or human subjects. A popular collection of measurements was taken by Gardner and Martin using a KEMAR dummy head, and made freely available [GM94, Gar98a]. Measurements of this kind are usually taken in an anechoic chamber, where a loudspeaker plays a test signal which approaches the head from the desired direction. The directions should be taken in such a way that two neighboring directions never exceed the localization blur, which ranges from about $\pm 3^\circ$ in azimuth for frontal sources, to about $\pm 20^\circ$ in elevation for sources above and slightly behind the listener [Bla83]. The test signal is usually a pseudo-random noise such as a Maximum-Length Sequence (MLS) [RV89] or a Golay code [ZGM92], which can easily be deconvolved from the measured response. The result of the measurements is a set of HRIRs that can be directly used as coefficients of a pair of FIR filters. Since the decay time of the HRIR is always less than a few milliseconds, 256 to 512 taps are sufficient at a sampling rate of 44.1 kHz.

A cookbook of HRIRs and direct convolution seems to be a viable solution to provide directionality to sound sources using today's technology. A fundamental limitation comes from the fact that HRIRs vary widely between different subjects, to such an extent that front-back reversals are fairly common when listening through someone else's HRIRs. Using individualized HRIRs dramatically improves the quality of localization. Moreover, since we unconsciously use small head movements to resolve possible directional ambiguities, head-motion tracking is also desirable.

There are several reasons that make a model of the external hearing system more desirable than a raw catalog of HRIRs. First of all, a model might be implemented more efficiently, thus allowing more sources to be spatialized in real time. Second, if the model is well understood, it might be described with a few parameters having a direct relationship with physical or geometric quantities. This latter possibility can save memory and allow easy calibration.

As happens with models for sound synthesis, we can try to model the effects or the causes of the modifications occurring on sound signals. The first approach consists in applying data reduction and filter design techniques, especially in the frequency domain, to the HRTFs. Much research has been devoted to discovering the amount of approximation that is tolerated by human listeners and how to design efficient IIR filters that implement the approximated HRTFs [KW92]. A recent experiment has shown that we are quite insensitive to the fine details of the HRTF magnitude spectrum, and that the lack of externalization often reported in prior studies might be due to incorrect coupling between the headphones and the ear canal [KC98]. Filter design techniques have been applied to the problem of approximating the desired HRTFs by means of low-order yet accurate linear systems. IIR filters of order as low as ten can be designed so that they keep enough spectral detail to allow good localization accuracy [MHV97]. Frequency warping has been proposed as a technique to increase the accuracy of approximations in the low frequency range [HZ99] by stretching the frequency axis according to the distribution of critical bands [ZF90]. One of the problems of the models based on signal pro-

cessing is that they do not increase our understanding of the underlying physical phenomena. As a consequence, it becomes difficult to control the parameters and we have to rely on collections of static configurations.

Modeling the structural properties of the system pinna-head-torso gives us the possibility of applying continuous variation to the positions of sound sources and to the morphology of the listener. Much of the physical/geometric properties can be understood by careful analysis of the HRIRs, plotted as surfaces, functions of the variables time and azimuth, or time and elevation. This is the approach taken by Brown and Duda [BD98] who came up with a model which can be structurally divided into three parts:

- Head Shadow and ITD
- Shoulder Echo
- Pinna Reflections

Starting from the approximation of the head as a rigid sphere that diffracts a plane wave, the shadowing effect can be effectively approximated by a first-order continuous-time system, i.e., a pole-zero couple in the Laplace complex plane:

$$s_z = \frac{-2\omega_0}{\alpha(\theta)} \quad (6.11)$$

$$s_p = -2\omega_0, \quad (6.12)$$

where ω_0 is related to the effective radius a of the head and the speed of sound c by

$$\omega_0 = \frac{c}{a}. \quad (6.13)$$

The position of the zero varies with the azimuth θ (see Fig. 6.9) according to the function

$$\alpha(\theta) = 1.05 + 0.95 \cos \left(\frac{\theta}{150^\circ} 180^\circ \right). \quad (6.14)$$

The pole-zero couple can be directly translated into a stable IIR digital filter by bilinear transformation [Mit98], and the resulting filter (with proper scaling) is

$$H_{\text{hs}} = \frac{(\omega_0 + \alpha F_s) + (\omega_0 - \alpha F_s)z^{-1}}{(\omega_0 + F_s) + (\omega_0 - F_s)z^{-1}}. \quad (6.15)$$

The ITD can be obtained by means of a first-order allpass filter [OS89, SF85] whose group delay in seconds is the following function of the azimuth angle θ :

$$\tau_h(\theta) = \begin{cases} -\frac{a}{c} \cos \theta & \text{if } 0 \leq |\theta| < \frac{\pi}{2} \\ \frac{a}{c} \left(|\theta| - \frac{\pi}{2} \right) & \text{if } \frac{\pi}{2} \leq |\theta| < \pi \end{cases}. \quad (6.16)$$

Actually, the group delay provided by the allpass filter varies with frequency, but for these purposes such variability can be neglected. Instead, the filter (6.15) gives

an excess delay at DC that is about 50 percent that given by (6.16). This increase in the group delay at DC is exactly what one observes for the real head [Kuh77], and it has already been outlined in Fig. 6.10. The overall magnitude and group delay responses of the block responsible for head shadowing and ITD are reported in Fig. 6.12. The M-file 6.3 implements the head-shadowing filter.

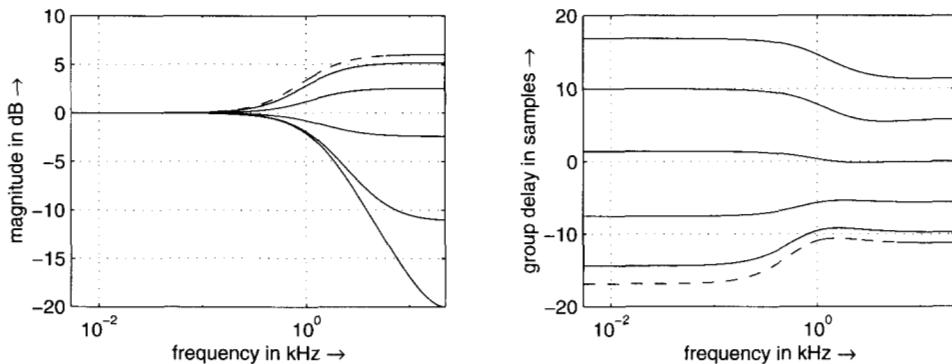


Figure 6.12 Magnitude and group delay responses of the block responsible for head shadowing and ITD ($f_s = 44100$ Hz). Azimuth ranging from 0 (dashed line) to π at steps of $\pi/6$.

M-file 6.3 (hsfilter.m)

```

function [output] = hsfilter(theta, Fs, input)
% hsfilter(theta, Fs, input)
%
% filters the input signal according to head shadowing
% theta is the angle with the frontal plane
% Fs is the sample rate

theta = theta + 90;
theta0 = 150 ; alfa_min = 0.05 ;
c = 334; % speed of sound
a = 0.08; % radius of head
w0 = c/a;
alfa = 1+ alfa_min/2 + (1- alfa_min/2)* cos(theta/ theta0* pi) ;

B = [(alfa+w0/Fs)/(1+w0/Fs), (-alfa+w0/Fs)/(1+w0/Fs)] ;
% numerator of Transfer Function
A = [1, -(1-w0/Fs)/(1+w0/Fs)] ;
% denominator of Transfer Function
if (abs(theta) < 90)
    gdelay = - Fs/w0*(cos(theta*pi/180) - 1)
else
    gdelay = Fs/w0*((abs(theta) - 90)*pi/180 + 1)
end

```

```

end;
a = (1 - gdelay)/(1 + gdelay);
% allpass filter coefficient
out_magn = filter(B, A, input);
output = filter([a, 1], [1, a], out_magn);

```

The function `hsfilter` has to be applied twice, for the left and right ears with opposite values of argument `theta`, to a given input sound.

In a rough approximation, the shoulder and torso effects are synthesized in a single echo. An approximate expression of the time delay can be deduced by the measurements reported in [BD98, Fig. 8]

$$\tau_{sh} = 1.2 \frac{180^\circ - \theta}{180^\circ} \left(1 - 0.00004 \left((\phi - 80^\circ) \frac{180^\circ}{180^\circ + \theta} \right)^2 \right) \text{ in ms ,} \quad (6.17)$$

and it is depicted in Fig. 6.13. The echo should also be attenuated as the source goes from frontal to lateral position.

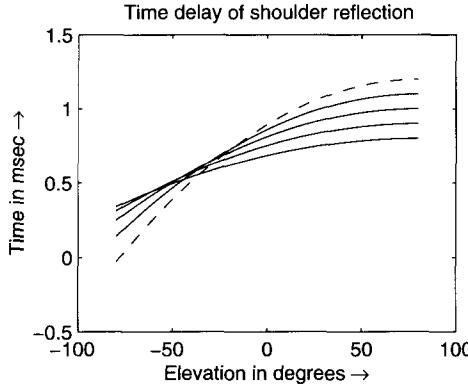


Figure 6.13 Delay time of the echo generated by the shoulders as a function of azimuth and elevation. Azimuth ranging from 0 (dashed line) to $\pi/3$ at steps of $\pi/12$.

Finally, the pinna provides multiple reflections that can be obtained by means of a tapped delay line. In the frequency domain, these short echoes translate into notches whose position is elevation dependent and that are frequently considered as the main cue for the perception of elevation [Ken95a]. In [BD98], a formula for the time delay of these echoes is given:

$$\tau_{pn} = A_n \cos(\theta/2) \sin(D_n(90^\circ - \phi)) + B_n . \quad (6.18)$$

The parameters are given in Table 6.1 together with the amplitude values ρ_{pn} of the reflections. The parameter D_n allows the adjustment of the model to the individual characteristics of the pinna, thus providing an effective knob for optimizing the localization properties of the model. Figure 6.14 depicts the delay time of the first

Table 6.1 Parameters for calculating amplitude and time delay of the reflections produced by the pinna model.

n	ρ_{pn}	A_n [samples]	B_n [samples]	D_n
2	0.5	1	2	$\cong 1$
3	-1	5	4	$\cong 0.5$
4	0.5	5	7	$\cong 0.5$
5	-0.25	5	11	$\cong 0.5$
6	0.25	5	13	$\cong 0.5$

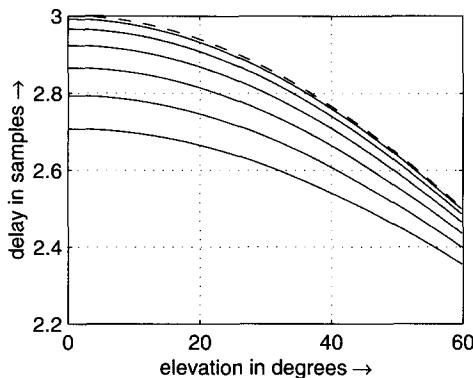


Figure 6.14 Delay time (in samples at $f_s = 44100$ Hz) of the first echo generated by the pinna as a function of azimuth and elevation. Azimuth ranging from 0 (dashed line) to $\pi/2$ at steps of $\pi/12$.

pinna echo (in samples at $f_s = 44100$ Hz) as a function of azimuth and elevation. The corresponding frequency notch lies somewhere between 7 and 10 kHz.

The structural model of the pinna-head-torso system is depicted in Fig. 6.15 with all its three functional blocks, repeated twice for the two ears. Even though the directional properties are retained by the model, anechoic sounds filtered by the model do not externalize well. As we have explained in section 6.3.3, there are several ways to improve the externalization in binaural listening.

Music Applications and Control

Several widely used software systems allow the possibility of spatializing sound sources for binaural listening. For instance, Tom Erbe's Soundhack, an award-winning software for the Macintosh, has a cookbook of HRTF coefficients and allows the musician to locate apparent sound sources in a 3D space. Similar operations can be carried out using the Csound language [BC00] and native opcodes.

We have already mentioned the IRCAM Spatialisateur as a powerful software

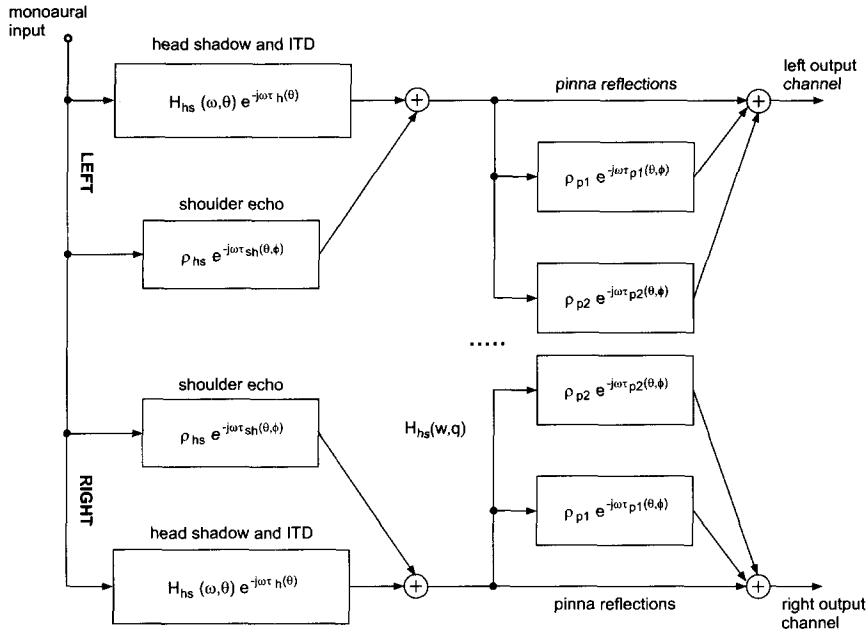


Figure 6.15 Structural model of the pinna-head-torso system.

spatialization system. It allows specification of a virtual room and virtual source trajectories in 3D space, and the rendering can be done either for loudspeakers or for headphones. In this latter case, HRTFs are used. The ability to provide two different output formats with a single system has been used in the production of the CD of Pierre Boulez's "Répons" [m-Bou84]. In fact, a few early and lucky customers could get another CD for free from the publishing company, with the binaural version of the same performance.

6.4 3D with Loudspeakers

6.4.1 Introduction

We outline three main approaches to sound spatialization by multi-loudspeaker layouts: holophonic reconstruction, transaural techniques, and methods relying on the precedence effect.

Holophonic reconstruction is the reproduction of a 2D or 3D soundfield in a confined area as a result of the interference of the wavefronts generated by different loudspeakers. Diffraction effects around the listener's head and torso, described in section 6.4.2, can be taken into account in order to optimize the rendering. We discuss two specific holophonic techniques, namely, 3D panning and Ambisonics, in sections 6.4.3 and 6.4.4.

Transaural spatialization is described in section 6.4.5 as a recasting of binaural techniques for presentations based on loudspeaker layouts.

The relative arrival time of the wavefronts from different loudspeakers can have a dramatic impact on the apparent source directions. A technique that introduces explicit delays among the loudspeakers is the room-within-the-room model, presented in section 6.4.6. This technique, even though less accurate, is less sensitive to changes of the listener position because it exploits the precedence effect.

6.4.2 Localization with Multiple Speakers

A listener facing a couple of loudspeakers receives the two signals x_{LL} and x_{RL} at the left ear, the first coming from the left loudspeaker, and the second coming from the right loudspeaker. Symmetrically, the two signals x_{RR} and x_{LR} are received at the right ear. If the right loudspeaker has a pure amplitude factor A_R and a pure time delay τ_R , a sinusoidal, unit-amplitude signal at the loudspeakers generates two sinusoidal signals at the ears. These signals are attenuated and shifted by the following complex weights:

$$\begin{aligned} x_L &= x_{LL} + x_{RL} = 1 + A_R A_H e^{-j\omega(\tau_R + \tau_H)} \\ x_R &= x_{RR} + x_{LR} = A_R e^{-j\omega\tau_R} + A_H e^{-j\omega\tau_H}, \end{aligned} \quad (6.19)$$

where A_H and τ_H are the amplitude factor and time delay given by the head transfer function to the contralateral ear from the direction of the loudspeaker. The situation is illustrated in Fig. 6.16. Since, in low frequency, A_H is almost unity, it can be seen from (6.19) with the help of Fig. 6.17 that a pure level difference (i.e. $\tau_R = 0$) at the loudspeakers generates a pure time delay at the ears. Conversely, a pure time delay between the loudspeakers generates a pure level difference at the ears. As is shown in [Bla83], the ear signal can even be stronger on the side of the lagging loudspeaker. The analysis of stationary signals becomes more complicated at higher frequencies, where the shadowing of the head cannot be ignored. In general, the time and level differences at the ears can give cues that are in contradiction to the time and level differences at the loudspeakers.

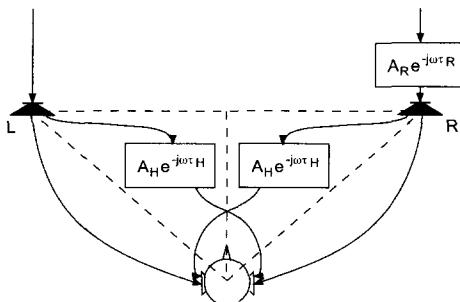


Figure 6.16 Transfer functions involved in a stereophonic layout.

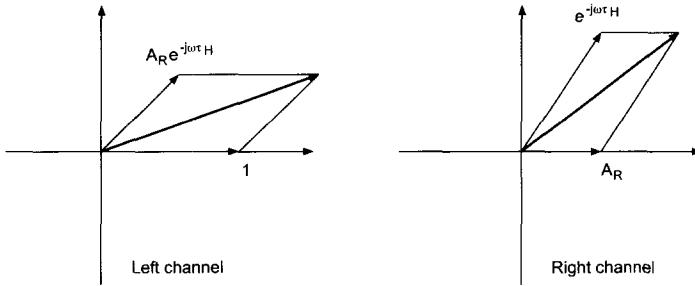


Figure 6.17 Vector illustration of the pure time difference (rotation of the bold vector) at the ears generated by a pure level difference at the loudspeakers.

Michael Gerzon developed some theories of localization in a sound field [Ger92a] that were somewhat validated by experiments and crafting experience. Gerzon called the meta-theory collecting all these theories “directional psychoacoustics”. The basic assumptions of directional psychoacoustics are:

- At low frequencies (up to about 500 Hz) the signals arriving at the two ears have a stable phase difference that is less than half a wavelength. The hearing system produces a sense of direction from this phase difference. The phase locking means that we can do a vector summation of the contributions from all loudspeakers in order to obtain a “velocity vector gain”

$$\mathbf{u} = \sum_i g_i \mathbf{l}_i \quad (6.20)$$

that is representative of the perceived direction in low frequency. In (6.20), g_i is the gain of the i -th loudspeaker and \mathbf{l}_i is the unit-length vector pointing from the loudspeaker to the listener (see Fig. 6.1).

- At high frequencies (from about 500 Hz up to about 3500 Hz) the signals arriving at the two ears are treated as incoherent, and the hearing system is mainly sensitive to the “energy vector gain”

$$\mathbf{e} = \sum_i |g_i|^2 \mathbf{l}_i . \quad (6.21)$$

The facts or assumptions provided by directional psychoacoustics are used to improve the quality of multichannel reproduction in several ways (see section 6.4.4).

6.4.3 3D Panning

The matrix-based approach used for stereo panning in section 6.2.1 can be generalized to an arbitrary number of loudspeakers located at any azimuth though nearly

equidistant from the listener. Such a generalization is called Vector Base Amplitude Panning (VBAP) [Pul97] and is based on a vector representation of positions in a Cartesian plane having its center in the position of the listener. The unit-magnitude vector pointing toward the virtual sound source \mathbf{u} can be expressed as a linear combination of the unit-magnitude column vectors \mathbf{l}_L and \mathbf{l}_R pointing toward the left and right loudspeakers, respectively. In matrix form, this combination can be expressed as

$$\mathbf{u} = \mathbf{L} \cdot \mathbf{g} = [\mathbf{l}_L \quad \mathbf{l}_R] \begin{bmatrix} g_L \\ g_R \end{bmatrix}. \quad (6.22)$$

Except for degenerate loudspeaker positions, the linear system of equations (6.22) can be solved in the vector of gains \mathbf{g} . This vector has not, in general, unit magnitude, but can be normalized by appropriate amplitude scaling. The solution of system (6.22) implies the inversion of matrix \mathbf{L} , but this can be done beforehand for a given loudspeaker configuration.

The generalization to more than two loudspeakers in a plane is obtained by considering, at any virtual source position, only one couple of loudspeakers, thus choosing the best vector base for that position.

The generalization to three dimensions is obtained by considering vector bases formed by three independent vectors in space. The vector of gains for such a 3D vector base is obtained by solving the system

$$\mathbf{u} = \mathbf{L} \cdot \mathbf{g} = [\mathbf{l}_L \quad \mathbf{l}_R \quad \mathbf{l}_Z] \begin{bmatrix} g_L \\ g_R \\ g_Z \end{bmatrix}. \quad (6.23)$$

Of course, having more than three loudspeakers in a 3D space implies, for any virtual source position, the selection of a local 3D vector base.

As indicated in [Pul97], VBAP ensures the maximum sharpness in sound source location. In fact:

- If the virtual sound source is located at a loudspeaker position, only that loudspeaker has nonzero gain;
- If the virtual sound source is located on a line connecting two loudspeakers, only those two loudspeakers have nonzero gain;
- If the virtual sound source is located on the triangle delimited by three adjacent loudspeakers, only those three loudspeakers have nonzero gain.

The formulation of VBAP given here is consistent with the low frequency formulation of directional psychoacoustics. The extension to high frequencies has also been proposed with the name Vector Base Panning (VBP) [PBJ98].

6.4.4 Ambisonics and Holophony

Ambisonics is a technique for spatial audio reproduction introduced in the early seventies by Michael Gerzon [Ger85, MM95]. While Vector Base Panning aims at projecting sound material into a 3D listening space, the focus of Ambisonics is really spatial recording, efficient encoding in a few audio channels, and reproduction by an appropriate loudspeaker set-up.

In Ambisonics the sound field is preferably encoded using the so-called B-format. 3D B-format is composed of four signals: W , X , Y , and Z , where W is a signal as taken from an omni-directional microphone, and X , Y , and Z are signals as taken from figure-of-eight microphones aligned with the orthogonal axes. If the four signals have to be produced starting from a monophonic sound signal s , the following encoding equations will apply:

$$\begin{aligned} X &= \cos \theta \cos \phi \cdot s \\ Y &= \sin \theta \cos \phi \cdot s \\ Z &= \sin \phi \cdot s \\ W &= \frac{\sqrt{2}}{2} \cdot s, \end{aligned} \quad (6.24)$$

where θ is the azimuth and ϕ is the elevation, as indicated in Fig. 6.9. The signal W is called the zero-order spherical harmonic of the sound field, and X , Y , and Z are called the first-order spherical harmonic components. The gain factors of the X , Y , and Z signals are the Cartesian coordinates of the unit length vector pointing to the virtual sound source.

The decoding stage will depend upon the actual loudspeaker layout. In the 3D case, the simplest layout is given by loudspeakers positioned at the corners of a cube. If the i -th loudspeaker is found along the direction of the unit vector \mathbf{l}_i , the corresponding gain is

$$g_i = \frac{1}{2} [G_1 W + G_2 [X \ Y \ Z] \cdot \mathbf{l}_i]. \quad (6.25)$$

As mentioned in section 6.4, some theories of directional psychoacoustics have been developed, mainly by Michael Gerzon [Ger92a], when to design the gains to be applied to the loudspeakers for setting the apparent direction of the sound source in a way consistent with our perception in a multichannel loudspeaker layout. These theories translate into different values of the gains G_1 and G_2 in (6.25). Ideally, these gains should be made frequency dependent, and replaced by filters whose shape can be carefully controlled [FU98].

If the i -th loudspeaker is aligned with one of the axes, the gain factors are the same as found in VBP, except for the zero order harmonic and for the scaling factor G_2 . In Ambisonics, if $G_1 = G_2 = 1$, when a sound source is located in the direction of a loudspeaker, the opposite loudspeaker has a null gain. This property is used to avoid antiphase signals from couples of loudspeakers, thus adding stability to the sound image. This is not necessary in VBP, since it uses only a local base of

loudspeakers and it does not treat the whole loudspeaker layout as a vector base. In this sense, Ambisonics is a global technique. The fact that VBP is a local panning method allows the use of arbitrary loudspeaker layouts. Extensions of Ambisonics to general layouts were also proposed by Gerzon [Ger92a], especially for applications in surround sound for home theater.

Ambisonics and VBP have been compared in terms of the direction and strength of the velocity and energy vectors [PBJ98], which are obtained by normalizing (6.20) and (6.21) to the total pressure gain and the total energy gain, respectively [Ger92a]. According to directional psychoacoustics these vectors should point in the same direction and should be as close as possible to one in order to provide a sharp sound image. The comparison showed that VBP outperforms Ambisonics for steady sound sources, as expected for the sharpness of a local panning. However, Ambisonics gives smoother transitions between loudspeakers, in such a way that it is more difficult to tell where the loudspeakers really are. The conclusions drawn by comparison of the vector gains have been confirmed by informal listening tests on real time implementations.

It should be pointed out that these kinds of assessments make some sense only if the following assumptions are met:

- the listener remains steady in the sweet spot
- the loudspeakers generate plane waves
- anechoic or free field listening.

In practice, these assumptions are rarely met and one should rely on assessment tools based on interference patterns emerging in a realistic model of the actual room and audience area [LS99].

Ambisonics can induce dramatic shifts in the apparent source position as the listener moves out of the sweet spot. For instance, the sound image can switch from front to back if we just move one step backward. For home theater applications, the need for sharper frontal images in Ambisonics was addressed by introduction of vector transformations on the encoded signals [GB98], thus giving a knob for controlling the forward dominance of sound distribution.

In the literature, techniques such as Holophony and Wave-Field Synthesis can be found. These are based on the mathematical property of analytic fields in an enclosure that make them describable as an integral over the boundary. If we can put infinitely many loudspeakers on a closed contour line, we can reproduce any pressure field in the plane area within the contour. In practice, with a finite number of loudspeakers, spatial aliasing puts a severe limit on accuracy. It has been shown [NE98] that Ambisonics is a special case of Holophony obtained for loudspeakers placed at infinity (i.e., generating plane waves), and that using a numerous circular array of microphones and loudspeakers is a feasible way of extending the sweet spot of accurate acoustic field reconstruction to something more than a square meter.

6.4.5 Transaural Stereo

If binaural audio material is played through a stereo loudspeaker layout, then almost any spatialization effect is likely to disappear. However, a relatively inexpensive signal processing apparatus, proposed in the early sixties by Schroeder [Sch61], can recreate the 3D listening experience at selected positions by preconditioning the signals feeding the loudspeakers. The audio display based on loudspeakers is often preferable because it is immune to fatigue and internalization problems that often arise with headphones.

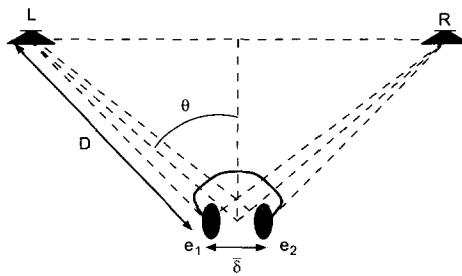


Figure 6.18 Binaural and transaural listening: geometry.

Assume that we want to recreate, by means of a couple of loudspeakers, the signals as they arrive to the ears from a couple of headphones. Referring to Fig. 6.18, $\bar{\delta}$ is the distance between the ears in spatial samples, i.e., the distance in meters multiplied by f_s/c , where c is the speed of sound and f_s is the sample rate, D is the distance in samples between a loudspeaker and the nearest ear, θ is the angle subtended by a loudspeaker with the median plane. Under the assumption of point-like loudspeakers, the excess distance from a loudspeaker to the contralateral ear produces an attenuation that can be approximated by

$$g \cong \frac{D}{\bar{\delta} \sin \theta + D}. \quad (6.26)$$

As we saw in section 6.3, the head of the listener introduces a shadowing effect which can be expressed by a lowpass transfer function $H(z)$. These considerations lead to the following matrix relationship between the signals at the ears ($e_1(z), e_2(z)$) and the signals at the loudspeakers ($L(z), R(z)$):

$$\begin{bmatrix} e_1(z) \\ e_2(z) \end{bmatrix} = \begin{bmatrix} 1 & gH(z)z^{-d} \\ gH(z)z^{-d} & 1 \end{bmatrix} \begin{bmatrix} L(z) \\ R(z) \end{bmatrix} \triangleq \mathbf{A}(z) \begin{bmatrix} L(z) \\ R(z) \end{bmatrix} \quad (6.27)$$

where d is the arrival time difference in samples of the signals at the ears. We should consider d a function of frequency, as in Fig. 6.10.a, but it is usually enough to set it to the low-frequency limit $d \cong 1.5\bar{\delta} \sin \theta$, easily derived from (6.8). The symmetric matrix \mathbf{A} can be easily inverted, thus giving

$$\mathbf{A}^{-1}(z) = \frac{1}{1 - g^2 H^2(z) z^{-2d}} \begin{bmatrix} 1 & -gH(z)z^{-d} \\ -gH(z)z^{-d} & 1 \end{bmatrix}. \quad (6.28)$$

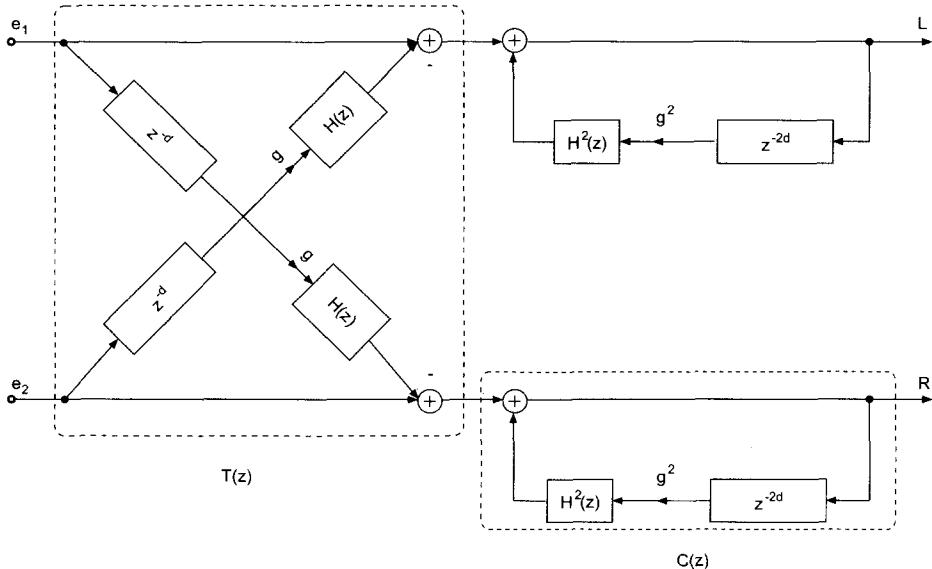


Figure 6.19 Binaural to transaural conversion.

The conversion from binaural to transaural is realized by the 2-input 2-output system represented by matrix (6.28) and depicted in Fig. 6.19. Here we have outlined two functional blocks: $T(z)$ is a lattice section, and $C(z)$ is a comb filter with a lowpass in the feedback loop. This decomposition is useful when cascading the transaural processor to a mono-to-binaural-stereo converter (see, e.g., section 6.6.1). In such a case, the linearity and time invariance of the functional blocks allow the use of a single comb filter before the mono-to-stereo converter. This equivalence is illustrated in Fig. 6.20. From a comparison of Figures 6.19 and 6.20 it is clear how the switch from transaural to binaural processing can be done simply by zeroing the coefficient g in both the lattice section and the comb filter.

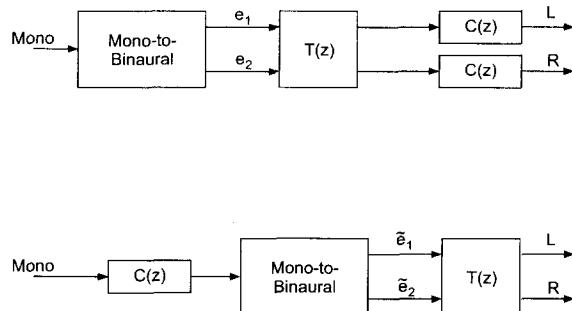


Figure 6.20 Mono-to-transaural-stereo converter.

Apparently, transaural stereo imposes strict requirements on the position of the listener and the absence of reflections in the room. However, a certain amount of head rotation and shift can be tolerated and, if the walls are not too close to the loudspeakers, the early room reflections arrive after the impulse response of the crosstalk canceller has vanished [CB89], so that its effectiveness is preserved. In his book [Gar98a], William Gardner presents different topologies and practical implementation details of transaural stereophonic processors, including a thorough physical and psychophysical validation.

6.4.6 Room-Within-the-Room Model

Panning and Ambisonics are methods for controlling the gains applied to the loudspeakers in order to approximate a target sound field at a privileged listener position. A completely different approach can be taken by controlling the relative time delay between the loudspeaker feeds. A model supporting this approach was introduced by Moore [Moo82], and can be described as a physical and geometric model. The metaphor underlying the Moore model is that of the room within a room, where the inner room has holes in the walls, corresponding to the positions of loudspeakers, and the outer room is the virtual room where sound events have to take place (Fig. 6.21). The simplest form of spatialization is obtained by drawing direct sound

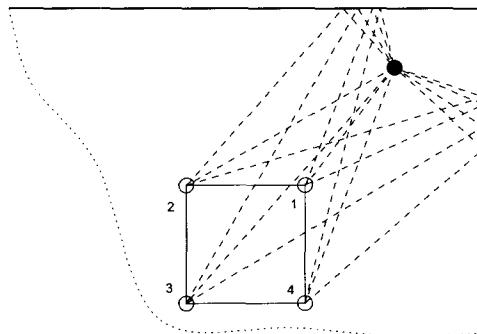


Figure 6.21 Moore's room in a room model.

rays from the virtual sound source to the holes of the inner room. If the outer room is anechoic, these are the only paths taken by sound waves to reach the inner room. The loudspeakers will be fed by signals delayed by an amount proportional to the length of these paths, and attenuated according to the relationship of inverse proportionality valid for propagation of spherical waves. In formulas, if l_i is the path length from the source to the i -th loudspeaker, and c is the speed of sound in air, the delay in seconds is set to

$$d_i = l_i/c , \quad (6.29)$$

and the gain is set to

$$g_i = \begin{cases} \frac{1}{l_i}, & l_i > 1 \\ 1, & l_i < 1 \end{cases} . \quad (6.30)$$

The formula for the amplitude gain is such that sources within the distance of 1m from the loudspeaker² will be connected to unity gain, thus avoiding the asymptotic divergence in amplitude implied by a point source of spherical waves.

The model is as accurate as the physical system being modeled would permit. A listener within a room would have a spatial perception of the outside soundscape whose accuracy will increase with the number of windows in the walls. Therefore, the perception becomes sharper by increasing the number of holes/loudspeakers. In reality, some of the holes will be masked by some walls, so that not all the rays will be effective³ (e.g. the rays to loudspeaker 3 in Fig. 6.21). In practice, the directional clarity of spatialization is increased if some form of directional panning is added to the base model, so that loudspeakers opposite the direction of the sound source are severely attenuated. In this case, it is not necessary to burden the model with an algorithm of ray-wall collision detection. The Pioneer Sound Field Controller is an example of 15-loudspeaker hemispherical array governed by these principles [Mar01].

A special case of the room in a room model is obtained when the loudspeakers are all located along a straight line, for instance above the listeners' heads. In this case we can think of holes in the ceiling of the inner room, and this layout is especially effective in reproducing movements of the virtual sound source along one dimension. Figure 6.22 illustrates this one-dimensional reduction of the model in the case of four loudspeakers and two listeners. From Fig. 6.22, it is easy to visualize the robustness of the method to different listener positions.

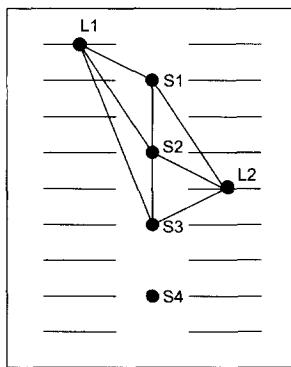


Figure 6.22 One-dimensional spatialization model.

In fact, the spatialization methods based on amplitude panning are generally designed for a tight listener position, and are overwhelmed by the precedence effect

²This distance is merely conventional.

³We are ignoring diffraction from this reasoning.

as soon as the listener is in a different position. The fact that most of the sound images collapse into one of the loudspeakers can be easily experienced by attending a pop music concert, or listening to a conventional car audio system. On the other hand, the Moore model relies explicitly on the precedence effect, in the sense that the sound image is biased toward the first loudspeaker being reached by the acoustic rays. As illustrated in Fig. 6.22 for one-dimensional spatialization, in general there is a wide area that is biased toward the same loudspeaker, just because the time delay of the virtual acoustic rays sums with the time delay of the actual paths connecting the loudspeakers with the listener. For instance, a virtual sound source located in the neighborhood of the loudspeaker S2 will be perceived as radiating from S2 by both listeners L1 and L2, even though the loudspeaker S1 is closer than S2 to L1. This is an intuitive explanation of the claim that the Moore model is able to provide consistent and robust spatialization to extended audiences [Moo82]. Another reason for robustness might be found in the fact that simultaneous level and time differences are applied to the loudspeakers. This has the effect of increasing the lateral displacement [Bla83] even for virtual sound sources such that the rays to different loudspeakers have similar lengths. Indeed, the localization of the sound source becomes even sharper if the level control is driven by laws that roll off more rapidly than the physical $1/d$ law of spherical waves.

An added benefit of the room within a room model is that the Doppler effect is intrinsically implemented. As the virtual sound source is moved in the outer room the delay lines representing the virtual rays change their lengths, thus producing the correct pitch shifts. It is true that different transpositions might affect different loudspeakers, as the variations are different for different rays, but this is consistent with the physical robustness of the technique. For instance, if in Fig. 6.22 the source is moving downwards starting from the middle of the room, the loudspeaker 2 will produce a downward transposition, and loudspeaker 4 will produce an upward transposition. Accordingly, the listeners located close to one of the loudspeakers will perceive the transposition that is most consistent with their position with respect to the virtual sound source.

The model of the room within a room works fine if the movements of the sound source are confined to a virtual space external to the inner room. This corresponds to an enlargement of the actual listening space and it is often a highly desirable situation. Moreover, it is natural to try to model the physical properties of the outer room, adding reflections at the walls and increasing the number of rays going from a sound source to the loudspeakers. This configuration, illustrated in Fig. 6.21 with first-order reflections, is a step from spatialization to reverberation and will be further discussed in section 6.5.

Music Applications and Control

Sound spatialization by means of loudspeakers has accompanied the history of electro-acoustic music in the second half of the twentieth century. An early spatialization system, the *potentiomètre d'espace* was used by J. Poullin to project Schaeffer's sounds of *musique concrète* into space [Pou57]. It is interesting that,

even in the fifties, the limitations of reproduction by loudspeakers, such as the inevitability of the sweet spot, and the difficult interplay between the actual listening room and the simulated space, were very clear [DiS98].

In section 6.2.5 we talked about spatialization in the music of Stockhausen since his influential work “*Gesang der Jünglinge*” [m-Sto56]. In the early realizations, the loudspeakers were treated as actual sources, and trajectories were programmed by detailed control of the relative gains of multiple audio channels. The idea of having virtual sound sources that are detached from the actual loudspeaker positions became popular after the works of John Chowning in the USA, and Hans Peter Haller in Europe. The former proposed a fully-computerized system for designing spatial trajectories. The latter designed an analog device, called a *Halaphon*, that allowed control of rotational trajectories in real time, by means of sliding potentiometers. The halaphon was used in many important compositions, for instance, early versions of “Répons” by Boulez [m-Bou84], or “Prometeo” [m-Non82] by Nono. Indeed, that device is especially important because it helped establish a practice of real-time performance by means of electronic devices (also called Live Electronics) [Hal95].

The layout of Fig. 6.22 was used in live electro-acoustic music performance to simulate sound wavefronts going back and forth above the audience [m-Pis95]. In this case, the Doppler shift affecting noisy sounds is a desirable side-effect of the spatialization system that magnifies the direction and velocity of movements. A two-dimensional implementation of the Moore spatialization model was used in a piece by Cifariello Ciardi [m-Cif95], where the main electro-acoustic gesture is that of the rolling metallic ball of a pinball being launched.

6.5 Reverberation

6.5.1 Acoustic and Perceptual Foundations

In the previous sections, our main concern has been the apparent positions of sound sources. The effects of the surrounding environment have been confined to changes in the perceived distance and position of the apparent source. In this section, we focus on sound reverberation as a natural phenomenon occurring when sound waves propagate in an enclosed space. Reverberation brings information about the nature and texture of materials, and about the size and shape of the room and of the objects inside it.

In order to analyze the various acoustical aspects of reverberation, we will consider a rectangular room containing an omni-directional point source. Other, more complicated and realistic situations can be considered to a large extent as a generalization of this simple case.

Point Source in an Enclosed Space

Consider the pressure wave generated by a small sphere pulsing at the radian frequency ω . The corresponding wave number is defined as $k = \omega/c$. The particle

velocity of air is radial, in phase with the pressure for large distances r ($r \gg 1/k$) and in phase quadrature with pressure for small distances ($r < 1/k$) [MI86]. At long distance from the source (far field), we approximate the condition of propagation of plane waves. On the other hand, near the source (near field), there is a large velocity component that, being in quadrature with pressure, is responsible for some reactive energy that does not radiate. A simplified analysis may conclude that, as soon as the sound waves have traveled for a sufficiently long path from the source, we can consider them as plane waves, thus leaving the more complicated formalism of spherical propagation. We will see that the plane wave description can be easily interpreted in the frequency domain and it allows a straightforward analysis of steady-state responses. In the proximity of the excitation event, we need descriptions based on spherical waves. Such descriptions are better carried on in the time domain.

Later on, we will introduce a third level of description, useful to account for “fine grain” phenomena that are not easily described by the two other approaches. Namely, we will introduce a particle description of sound, which we will take advantage of when describing the diffusion due to rough walls.

Frequency Domain: Normal Modes

The acoustics of an enclosed space can be described as a superposition of normal modes, which are standing waves that can be set up in the gas filling the box. Such a description can be derived analytically for simple shapes, such as the rectangular one, and it represents a powerful tool for understanding the behavior of the room in the frequency domain.

It is easier to calculate the normal modes under the assumption that the sound waves are propagating in the far field and after the end of the initial transient. This assumption allows consideration of all the wavefronts as planes. The normal modes of a rectangular room having sizes $[l_x, l_y, l_z]$ can be derived by forcing a plane-wave solution into the 3D wave equation and setting the particle velocity to zero at the boundary [MI86]. As a result, we get the frequencies of the normal modes

$$f = \frac{c}{2} \left[\left(\frac{n_x}{l_x} \right)^2 + \left(\frac{n_y}{l_y} \right)^2 + \left(\frac{n_z}{l_z} \right)^2 \right]^{\frac{1}{2}}, \quad (6.31)$$

where

$$\mathbf{n} = [n_x, n_y, n_z] \quad (6.32)$$

is a triplet of non-negative integer numbers characterizing the normal mode.

Each normal mode is supported by a plane wave propagating in the direction represented by the vector

$$\mathbf{v} = \left[\frac{n_x}{l_x}, \frac{n_y}{l_y}, \frac{n_z}{l_z} \right]. \quad (6.33)$$

It is clear from (6.31) and (6.33) that a direction is associated with the modes having frequencies that are multiples of the same fundamental. In other words, all the triplets that are multiples of an irreducible triplet are associated with a harmonic series of resonant frequencies with the same direction in space. This property suggests that a harmonic series of normal frequencies can be reproduced by means of a comb filter, i.e. by means of a feedback delay line whose length is

$$d = \frac{1}{f_0}, \quad (6.34)$$

where f_0 is the fundamental of the harmonic series.

The collection of a normal mode and all its multiples can be thought of as a plane wave bouncing back and forth in the enclosure. In order for an enclosure having finite extent to support an infinite plane wavefront, it is necessary to bend the wavefront at the walls in such a way that it forms a closed constant-area surface. This interpretation can be visualized in two dimensions by means of plane wave loops having constant length (see Fig. 6.23). In this representation it is easy to verify that the time interval d is the time lag between two consecutive collisions of plane wavefronts along the diagonal.

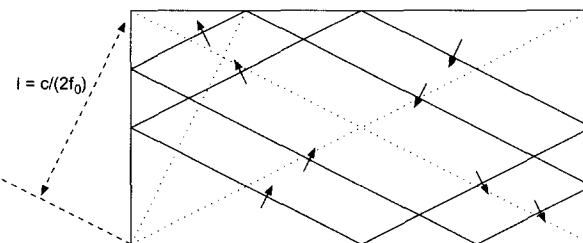


Figure 6.23 Plane wave loops for the normal mode $n_x = 1, n_y = 1$.

Not all the modes are excited with the same intensity in all positions. For instance, in the central point of a rectangular box, only one-eighth of the modes get excited, i.e. only those modes having all even components in the triplet \mathbf{n} , as the other modes have a nodal point in the middle. The most intense excitation is obtained at the corners of the room.

Time Domain: Acoustic Rays

Another kind of description of room acoustics considers the propagation of acoustic rays, i.e. portions of spherical waves having small aperture. This is the approach taken in geometrical acoustics, which is analogous to geometrical optics for light waves.

Some basic assumptions have to be made in order to ensure the validity of description by geometric rays. The first assumption is that the wavelengths interested by propagation are much smaller than the finest geometric description of the room.

The second assumption is that we can ignore all the phenomena of diffraction and interference. The first assumption is not very restrictive, since acoustic rays are used to represent events in a short time-scale, i.e. in a very large frequency range, and diffraction is also negligible at high frequencies. The absence of interference is also verified in most practical cases, where rays are mutually incoherent.

Along an acoustic ray, the pressure decreases as the reciprocal of distance, since the energy conveyed by a spherical wavefront has to be conserved during propagation, and the area occupied by the front increases as the square of distance from the source.

When a ray hits a surface, it is reflected in a way that depends on the nature of the surface. Smooth walls produce mirror reflections, i.e. the angle θ formed by the incoming ray with the normal to the surface is equal to the angle formed by the outgoing ray with the same normal. Moreover, the incoming ray, the outgoing ray, and the normal, all lie on the same plane.

Usually there is some filtering associated with a reflection, due to the absorbing properties of the wall material. In general, there is a frequency-dependent attenuation and a non constant time delay. Such filtering can be represented by a complex reflection function R relating the outgoing and incoming pressure wavefronts. R is dependent on the angle θ according to the formula [Kut91]

$$R = \frac{Z \cos \theta - 1}{Z \cos \theta + 1}, \quad (6.35)$$

where Z is the characteristic impedance of the wall. If the wall is smooth and rigid Z approaches infinity. In this case we say that the reflection is perfect, and the reflection function R is unitary at all frequencies and for any angle.

There are two popular approaches to compute acoustic rays for room acoustics: ray tracing and the image method. Ray tracing has been very popular in computer graphics since the eighties as a technique for producing realistic images. In graphics, an image is constructed by tracing light rays back from the eye (or the focal plane) to the light sources. In audio, an acoustic image is obtained by tracing acoustic rays back from the ear (or the microphone points) to the sound sources. In both cases, the scene is traversed by the rays and all the reflections are correctly reproduced. An important difference between sounds and images is that, while in graphics the light can be considered as propagated in all the points of the scene in a frame sample, in audio the sample rate is much higher and the speed of propagation is much smaller. As a consequence, each ray has to be associated with a delay line, thus adjoining memory occupation and computational complexity to a method that is already quite time-consuming.

The image method [AB79] is trivial when explained by means of a 2-D medium (e.g. a rectangular membrane). In Fig. 6.24 a source S is reflected symmetrically at the boundaries, and the source images are reflected about the images of the boundaries. The resulting tessellation of the plane is such that every segment connecting an image of the source with a receiving point R corresponds to one unique path connecting the source to the receiver after a certain number of specular reflections.

This method is easily extended to 3-D and, with many complications and increase in complexity, to rooms described as arbitrary polyhedra [Bor84].

Both the ray tracing technique and the image method are mainly used by architects in acoustic Computer Aided Design, where there is no special concern for real-time performance. On the other hand, in these applications the accuracy of results is particularly important in predicting the acoustic quality of a hall before it is constructed.

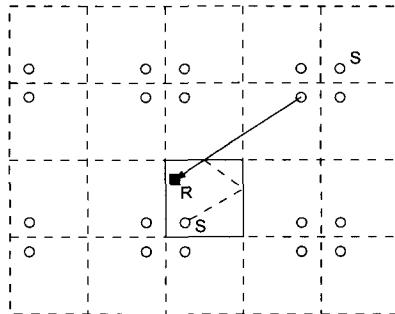


Figure 6.24 The image method in 2-D.

Sounds as Particles: Diffusion

Up to this point we have discussed room acoustics at the level of macroscopic phenomena, i.e. rays and normal modes. If the target is complexity (or realism) at the final result, we need to consider also the microscopic phenomena occurring in a physical environment. The most important of these second-order phenomena is diffusion of sound waves due to the fine grain geometry of the enclosure. Diffusion is better described using a particle representation of sound. In computer graphics a similar representation is used for modeling surface light diffusion and shadows. In order to adapt the same tools to acoustics, we need to keep the assumption of broadband signals. Moreover, we are interested in local interactions with surfaces having geometric irregularities (roughness) that are comparable with the wavelengths we are dealing with. Still, we need to assume that the acoustic particles are mutually incoherent, in order to avoid cancellations between particles. Once these hypotheses are assumed to be valid, it is natural to consider a sound particle as any brief acoustic signal having a broad frequency extension.

The diffusion model that is commonly used in computer graphics [CW93] can be readily adapted to the acoustic domain [Roc96], so that we can use the same symbols and terminology. In particular, diffusion at a certain point of the enclosure is described by a Bidirectional Reflection Distribution Function (BRDF) $f_r(\psi_i, \psi_o)$, which is the ratio between the radiance reflected along direction ψ_o and the irradiance incident along direction ψ_i .

The diffusion is said to be Lambertian when the BRDF is constant. In this case the sound particles are diffused in any direction with the same probability,

regardless of the incident direction. Vice versa, if all the sound particles arriving from direction ψ_i are redirected to the specular direction $\psi_o = -\psi_i$ we have a mirror reflection, i.e. no diffusion at all. Actual walls are always somewhere in between a mirror reflector and a Lambertian diffusor.

Quantities such as those defined in this section have been useful in improving the accuracy of estimates of the reverberation time of real rooms [Kut95].

Objective and Subjective Attributes of Room Acoustics

A very short sound pulse, such as a gun shot, generates a reverberating tail that has in most cases an exponentially decaying shape. The fundamental objective attribute of room acoustics is the reverberation time, defined by W.C. Sabine as the time interval in which the reverberation level drops down by 60 dB.

Extensive experimentation by computer music researchers showed that a very good, if not the best, impulse response for reverberation is obtained by generating Gaussian white noise and enveloping it with an exponential decay [Moo79]. The actual reverberated sound can be obtained by convolution, even though this operation is computationally expensive. Any frequency-dependent attenuation can be introduced by pre-filtering the impulse response. A finer control of the quality of reverberation can be exerted by modification of “perceptual factors”, which are subjective attributes that have been extracted from psychophysical experimentation. Research in this field proceeds by extracting scales of subjective acoustical experience and by correlating them with physical parameters. Most of the studies conducted up to 1992 were summarized in [Ber92].

Research at IRCAM in Paris [Jul95] tried to provide a minimal set of independent parameters that give an exhaustive characterization of room acoustic quality. These parameters are divided into three categories [Jot99]:

1. Source perception (related to the spectrum and relative energy of direct sound and early reflections):
 - presence (ratio between direct sound and early reverberated energy)
 - brilliance (high frequency variation of early reverberated energy)
 - warmth (low frequency variation of early reverberated energy)
2. Source/Room interaction (related to the relative energies of direct sound, early and late reflections, and to the early decay time):
 - envelopment (energy of early reflections relative to direct sound)
 - room presence (energy of late reverberated sound)
 - running reverberance (early decay time of the room impulse response)
3. Room perception (related to the late decay time and to its frequency variations):

- late reverberance (late decay time of the room impulse response)
- heaviness (low frequency variation of decay time)
- liveness (high frequency variation of decay time).

In a model that provides these parameters as knobs in a “perceptual interface”, the “presence” parameter can be controlled to give an impression of distance of the sound source, in a way similar to what we showed in section 6.2.3, and the “envelopment” parameter can be adjusted to control the impression of being surrounded by sound. Some of the parameters are perceived as timbral colorations while the source is playing (running reverberance), while some others are perceived from the sound tails left in pauses between sound events (late reverberance). Griesinger [Gri99] gives a definition of these two quantities that is based on slicing the room impulse response into segments of 160 ms each. Running reverberation, “the amount of self support one hears while playing”, is defined as the ratio between the energies in the second and first segments of the impulse response, and it is a measure of reverberant loudness that is independent of reverberation time. A room impulse response with a reverberation time of 0.5 s can sound just as loud (in terms of running reverberance) as a room with a reverberation time of 2 s [Gri94]. The running reverberance is an important parameter of room acoustics and its preferred value depends on the instrument and on the music genre [Gri94], being low for speech and relatively high for orchestral romantic music. Therefore, in artificial reverberation it is useful to have a knob that allows adjustment of the running reverberance to fit the music that is being played. Most of the times, the goal is to make reverberation audible but not overwhelming.

Envelopment is a controversial attribute, that is difficult to formalize but is known to be a key component of a pleasant source/room combination. The most recent investigations tend to relate envelopment to fluctuations in ITD and IID due to the spatial perception of early reflections. Therefore, to appreciate envelopment the early reflections should be appropriately spatialized. Blauert and Lindemann [BL86] showed that lateral reflections below 3 kHz contribute to a sense of depth, while higher frequency components contribute to a sense of surround. However, since lateral reflections produce ITD fluctuations and ITD is direction and frequency dependent, the most effective angle for a lateral reflection depends on frequency. Low frequency reflections produce the largest fluctuations if they come from lateral directions that are perpendicular to the median plane, while higher frequencies are more effective if they are closer to the median plane [Gri97]. Griesinger [Gri97, Gri99] developed a theory of spatial impression (another term meaning envelopment) that puts this multifaceted attribute of room acoustics into a cognitive framework where it is related to other phenomena such as the precedence effect. Namely, there is a “continuous spatial impression” that is perceived when lateral reflections are added to the perception of continuous sounds. Abrupt changes, which are always accompanied by the precedence effect, give rise to an “early spatial impression” by means of lateral reflections coming within 50 ms of the sound event, and to a “background spatial impression” by means of spatialized late reverberation. This latter form of spatial impression is considered to be the most important for the overall perception

of envelopment, as it is perceptually segregated from the streams of sound events that form a sonic foreground. Envelopment is considered to be a desirable feature of large rooms, and is absent in small rooms. However, sometimes envelopment is confused with the apparent source width, which is also an attribute of the source/room combination, but this can be high even in small rooms. The apparent source width will be further discussed in section 6.6.1.

6.5.2 Classic Reverberation Tools

In the second half of the twentieth century, several engineers and acousticians tried to invent electronic devices capable of simulating the long-term effects of sound propagation in enclosures. The most important pioneering work in the field of artificial reverberation has been that of Manfred Schroeder at the Bell Laboratories in the early sixties [Sch61, Sch62, Sch70, Sch73, SL61]. Schroeder introduced the recursive comb filters and the delay-based allpass filters as computational structures suitable for the inexpensive simulation of complex patterns of echoes. In particular, the allpass filter based on the recursive delay line has the form

$$y(n) = -g \cdot x(n) + x(n-m) + g \cdot y(n-m), \quad (6.36)$$

where m is the length of the delay in samples. The filter structure is depicted in Fig. 6.25, where $A(z)$ is usually replaced by a delay line. This filter allows a dense impulse response and a flat frequency response to be obtained. Such a structure rapidly became a standard component used in almost all the artificial reverberators designed until nowadays [Moo79]. It is usually assumed that the allpass filters do not introduce coloration in the input sound. However, this assumption is valid from a perceptual viewpoint only if the delay line is much shorter than the integration time of the ear, i.e. about 50 ms [ZF90]. If this is not the case, the time-domain effects become much more relevant and the timbre of the incoming signal is significantly affected.

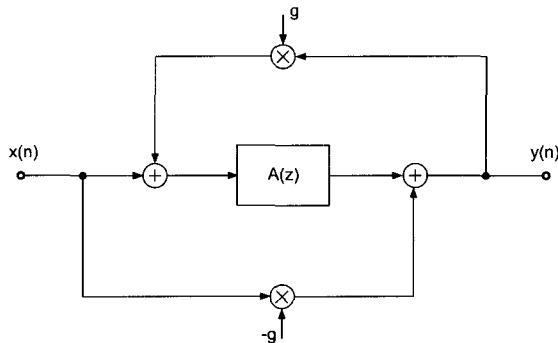


Figure 6.25 The allpass filter structure.

In the seventies, Michael Gerzon generalized the single-input single-output allpass filter to a multi-input multi-output structure, where the delay line of m samples

has been replaced by a order- N unitary network [Ger76]. Examples of trivial unitary networks are orthogonal matrices, and parallel connections of delay lines or allpass filters. The idea behind this generalization is that of increasing the complexity of the impulse response without introducing appreciable coloration in frequency. According to Gerzon's generalization, allpass filters can be nested within allpass structures, in a telescopic fashion. Such embedding is shown to be equivalent to lattice allpass structures [Gar98b], and it is realizable as long as there is at least one delay element in the block $A(z)$ of Fig. 6.25.

An extensive experimentation on structures for artificial reverberation was conducted by Andy Moorer in the late seventies [Moo79]. He extended the work done by Schroeder [Sch70] in relating some basic computational structures (e.g., tapped delay lines, comb and allpass filters) with the physical behavior of actual rooms. In particular, it was noticed that the early reflections have great importance in the perception of the acoustic space, and that a direct-form FIR filter can reproduce these early reflections explicitly and accurately. Usually this FIR filter is implemented as a tapped delay line, i.e. a delay line with multiple reading points that are weighted and summed together to provide a single output. This output signal feeds, in Moorer's architecture, a series of allpass filters and a parallel group of comb filters. Another improvement introduced by Moorer was the replacement of the simple gain of feedback delay lines in comb filters with lowpass filters resembling the effects of air absorption and lossy reflections.

An original approach to reverberation was taken by Julius Smith in 1985, when he proposed the Digital Waveguide Networks (DWN) as a viable starting point for the design of numerical reverberators [Smi85]. The idea of Waveguide Reverberators is that of building a network of waveguide branches (i.e., bidirectional delay lines simulating wave propagation in a duct or a string) capable of producing the desired early reflections and a diffuse, sufficiently dense reverb. If the network is augmented with lowpass filters it is possible to shape the decay time with frequency. In other words, waveguide reverberators are built in two steps: the first step is the construction of a prototype lossless network, the second step is the introduction of the desired amount of losses. This procedure ensures good numerical properties and a good control over stability [Smi86, Vai93]. In ideal terms, the quality of a prototype lossless reverberator is evaluated with respect to the whiteness and smoothness of the noise that is generated as response to an impulse. The fine control of decay time at different frequencies is decoupled from the structural aspects of the reverberator.

Among the classic reverberation tools we should also mention the structures proposed by Stautner and Puckette [SP82], and by Jot [Jot92]. These structures form the basis of the Feedback Delay Networks, which will be discussed in greater detail in section 6.5.3.

Clusters of Comb/Allpass Filters

The construction of high-quality reverberators is half an art and half a science. Several structures and many parameterizations were proposed in the past, especially in non-disclosed form within commercial reverb units [Dat97]. In most cases,

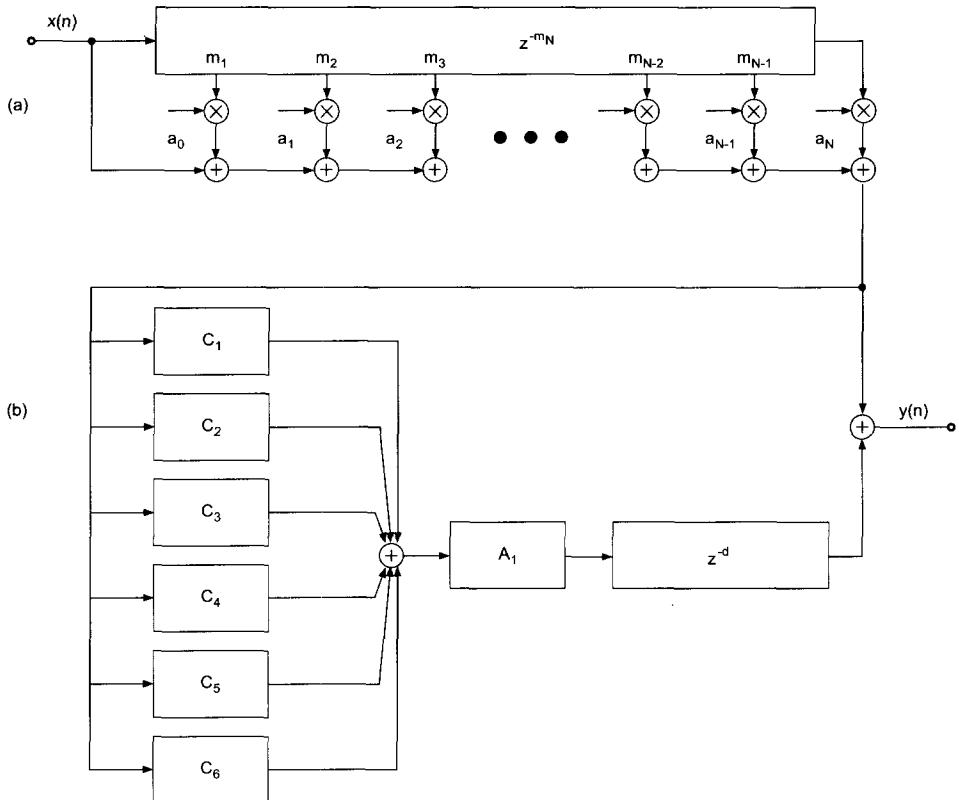


Figure 6.26 Moorer's reverberator.

the various structures are combinations of comb and allpass elementary blocks, as suggested by Schroeder in his early works. As an example, we briefly describe the Moorer's preferred structure [Moo79], depicted in Fig. 6.26. The block (a) of the Moorer's reverb takes care of the early reflections by means of a tapped delay line. The resulting signal is forwarded to the block (b), which is the parallel of a direct path on one branch, and a delayed, attenuated diffuse reverberator on the other branch. The output of the reverberator is delayed in such a way that the last of the early echoes coming out of block (a) reaches the output before the first of the non-null samples coming out of the diffuse reverberator. In Moorer's preferred implementation, the reverberator of block (b) is best implemented as a parallel group of six comb filters, each with a first-order lowpass filter in the loop, and a single allpass filter. In [Moo79], it is suggested setting the allpass delay length to 6 ms and the allpass coefficient to 0.7. Despite the fact that any allpass filter does not add coloration in the magnitude frequency response, its time response can give a metallic character to the sound, or add some unwanted roughness and granularity. The feedback attenuation coefficients and the lowpass filters of the comb filters can be tuned to resemble a realistic and smooth decay. In particular, the attenuation

coefficients g_i determine the overall decay time of the series of echoes generated by each comb filter. If the desired decay time (usually defined for an attenuation level of 60 dB) is T_d , the gain of each comb filter has to be set to

$$g_i = 10^{-3 \frac{T_d f_s}{m_i}}, \quad (6.37)$$

where f_s is the sampling rate and m_i is the delay length in samples. Further attenuation at high frequencies is provided by the feedback lowpass filters, whose coefficient can also be related with decay time at a specific frequency or fine tuned by direct experimentation. In [Moo79], an example set of feedback attenuation and allpass coefficients is provided, together with some suggested values for the delay lengths of the comb filters. As a general rule, they should be distributed over a ratio 1 : 1.5 between 50 and 80 ms. Schroeder suggested a number-theoretic criterion for a more precise choice of the delay lengths [Sch73]: the lengths in samples should be mutually coprime (or incommensurate) to reduce the superimposition of echoes in the impulse response, thus reducing the so called flutter echoes. This same criterion might be applied to the distances between each echo and the direct sound in early reflections. However, as it was noticed by Moorer [Moo79], the results are usually better if the taps are positioned according to the reflections computed by means of some geometric modeling technique, such as the image method. As will be explained in section 6.5.3, even the lengths of the recirculating delays can be computed from the geometric analysis of the normal modes of actual room shapes.

6.5.3 Feedback Delay Networks

In 1982, J. Stautner and M. Puckette [SP82] introduced a structure for artificial reverberation based on delay lines interconnected in a feedback loop by means of a matrix (see Fig. 6.27). More recently, structures such as this have been called Feedback Delay Networks (FDN). The Stautner and Puckette FDN was obtained as a vector generalization of the recursive comb filter

$$y(n) = x(n - m) + g \cdot y(n - m), \quad (6.38)$$

where the m -sample delay line was replaced by a bunch of delay lines of different lengths, and the feedback gain g was replaced by a feedback matrix \mathbf{G} . Stautner and Puckette proposed the following feedback matrix:

$$\mathbf{G} = \frac{g}{\sqrt{2}} \begin{bmatrix} 0 & 1 & 1 & 0 \\ -1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix}. \quad (6.39)$$

Due to its sparse special structure, \mathbf{G} requires only one multiply per output channel.

More recently, Jean-Marc Jot has investigated the possibilities of FDNs very thoroughly. He proposed using some classes of unitary matrices allowing efficient implementation. Moreover, he showed how to control the positions of the poles of

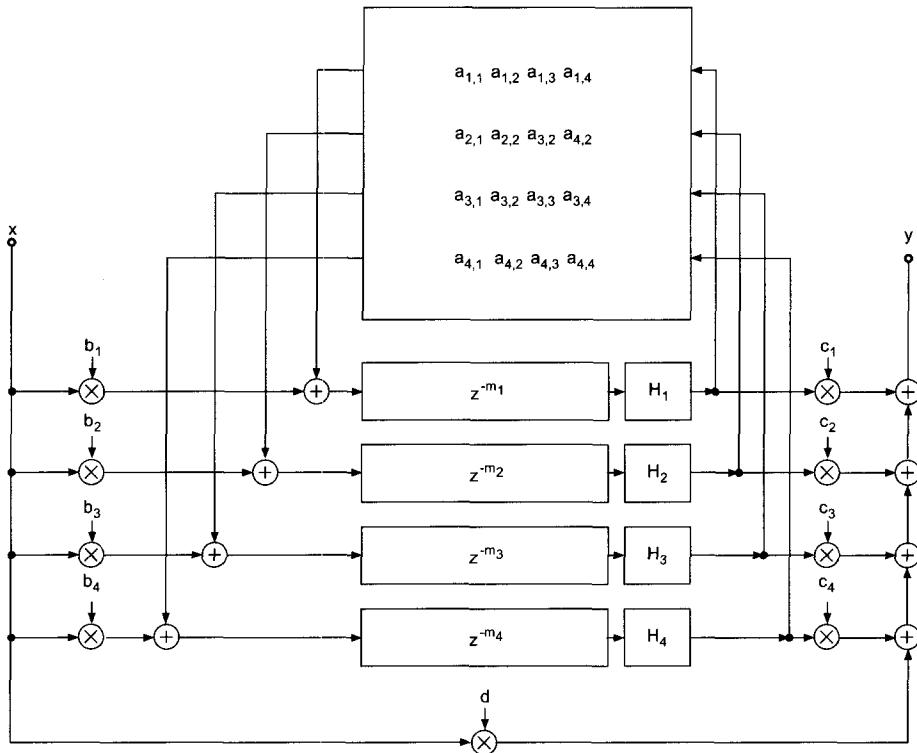


Figure 6.27 Fourth-order feedback delay network.

the structure in order to impose a desired decay time at various frequencies [Jot92]. His considerations were driven by perceptual criteria and the general goal is to obtain an ideal diffuse reverb. In this context, Jot introduced the important design criterion that all the modes of a frequency neighborhood should decay at the same rate, in order to avoid the persistence of isolated, ringing resonances in the tail of the reverb [JC91]. This is not what happens in real rooms though, where different modes of close resonance frequencies can be differently affected by wall absorption [Mor91]. However, it is generally believed that the slow variation of decay rates with frequency produces smooth and pleasant impulse responses.

General Structure

Referring to Fig. 6.27, an FDN is built starting from N delay lines, each being $\tau_i = m_i T_s$ seconds long, where $T_s = 1/f_s$ is the sampling interval. The FDN is

completely described by the following equations:

$$\begin{aligned} y(n) &= \sum_{i=1}^N c_i s_i(n) + d x(n) \\ s_i(n+m_i) &= \sum_{j=1}^N a_{i,j} s_j(n) + b_i x(n) \end{aligned} \quad (6.40)$$

where $s_i(n)$, $1 \leq i \leq N$, are the delay outputs at the n -th time sample. If $m_i = 1$ for every i , we obtain the well-known state space description of a discrete-time linear system [Kai80]. In the case of FDNs, m_i are typically numbers of the orders of hundreds or thousands, and the variables $s_i(n)$ are only a small subset of the system state at time n , being the whole state represented by the content of all the delay lines.

From the state-variable description of the FDN, it is possible to find the system transfer function [Roc96, RS97] as

$$H(z) = \frac{Y(z)}{X(z)} = \mathbf{c}^T [\mathbf{D}(z^{-1}) - \mathbf{A}]^{-1} \mathbf{b} + d. \quad (6.41)$$

The diagonal matrix $\mathbf{D}(z) = \text{diag}(z^{-m_1}, z^{-m_2}, \dots, z^{-m_N})$ is called the delay matrix, and $\mathbf{A} = [a_{i,j}]_{N \times N}$ is called the feedback matrix.

The stability properties of a FDN are all ascribed to the feedback matrix. The fact that $\|\mathbf{A}\|^n$ decays exponentially with n ensures that the whole structure is stable [Roc96, RS97].

The poles of the FDN are found as the solutions of

$$\det[\mathbf{A} - \mathbf{D}(z^{-1})] = 0. \quad (6.42)$$

In order to have all the poles on the unit circle, it is sufficient to choose a unitary matrix. This choice leads to the construction of a lossless prototype but this is not the only choice allowed.

The zeros of the transfer function can also be found [Roc96, RS97] as the solutions of

$$\det[\mathbf{A} - \mathbf{b} \frac{1}{d} \mathbf{c}^T - \mathbf{D}(z^{-1})] = 0. \quad (6.43)$$

In practice, once we have constructed a lossless FDN prototype, we must insert attenuation coefficients and filters in the feedback loop. For instance, following the indications of Jot [JC91], we can cascade every delay line with a gain

$$g_i = \alpha^{m_i}. \quad (6.44)$$

This corresponds to replacing $D(z)$ with $D(z/\alpha)$ in (6.41). With this choice of the attenuation coefficients, all the poles are contracted by the same factor α . As a

consequence, all the modes decay with the same rate, and the reverberation time (defined for a level attenuation of 60 dB) is given by

$$T_d = \frac{-3T_s}{\log \alpha} . \quad (6.45)$$

In order to have a faster decay at higher frequencies, as happens in real enclosures, we must cascade the delay lines with lowpass filters. If the attenuation coefficients g_i are replaced by lowpass filters, we can still get a local smoothness of decay times at various frequencies by satisfying the condition (6.44), where g_i and α have been made frequency dependent:

$$G_i(z) = A^{m_i}(z), \quad (6.46)$$

where $A(z)$ can be interpreted as per-sample filtering [JS83, JC91, Smi92].

It is important to note that a uniform decay of neighboring modes, even though commonly desired in artificial reverberation, is not found in real enclosures. The normal modes of a room are associated with stationary waves, whose absorption depends on the spatial directions taken by these waves. For instance, in a rectangular enclosure, axial waves are absorbed less than oblique waves [Mor91]. Therefore, neighboring modes associated with different directions can have different reverberation times. Actually, for commonly found rooms having irregularities in the geometry and in the materials, the response is close to that of a room having diffusive walls, where the energy rapidly spreads among the different modes. In these cases, we can find that the decay time is quite uniform among the modes [Kut95].

Parameterization

The main questions arising once we have established a computational structure called FDN are: What are the numbers that can be put in place of the many coefficients of the structure? How should these numbers be chosen?

The most delicate part of the structure is the feedback matrix. In fact, it governs the stability of the whole structure. In particular, it is desirable to start with a lossless prototype, i.e. a reference structure providing an endless, flat decay. The reader interested in general matrix classes that might work as prototypes is referred to the literature [Jot92, RS97, Roc97, Gar98b]. Here we only mention the class of circulant matrices, having the general form

$$\mathbf{A} = \begin{bmatrix} a(0) & a(1) & \dots & a(N-1) \\ a(N-1) & a(0) & \dots & a(N-2) \\ \dots & & & \\ a(1) & \dots & a(N-1) & a(0) \end{bmatrix} .$$

A matrix such as this is used in the Csound **babo** opcode. The stability of a FDN is related to the magnitude of its eigenvalues, which can be computed by the Discrete Fourier Transform of the first row, in the case of a circulant matrix. By keeping

these eigenvalues on the unit circle (i.e., magnitude one) we ensure that the whole structure is stable and lossless. The control over the angle of the eigenvalues can be translated into a direct control over the degree of diffusion of the enclosure that is being simulated by the FDN. The limiting cases are the diagonal matrix, corresponding to perfectly reflecting walls, and the matrix whose rows are sequences of equal-magnitude numbers and (pseudo-)randomly distributed signs [Roc97].

Another critical set of parameters is given by the lengths of the delay lines. Several authors suggested the use of sample lengths that are mutually coprime numbers in order to minimize the collision of echoes in the impulse response. However, if the FDN is linked to a physical and geometrical interpretation, as it is done in the Ball-within-the-Box (BaBo) model [Roc95], the delay lengths are derived from the geometry of the room being simulated and the resulting digital reverb quality is related to the quality of the actual room. How such derivation of delay lengths is actually performed is understandable from Fig. 6.23. A delay line will be associated to a harmonic series of normal modes, all obtainable from a plane wave loop that bounces back and forth within the enclosure. The delay length for the particular series of normal modes represented in Fig. 6.23 is given by the time interval between two consecutive collisions of the plane wavefront along the main diagonal, i.e. twice the time taken to travel the distance

$$l = \frac{c}{2f_0} , \quad (6.47)$$

being f_0 the fundamental frequency of the harmonic modal series. The extension of the BaBo model to spherical enclosures was presented in [RD01].

6.5.4 Convolution with Room Impulse Responses

If the impulse response of a target room is readily available, the most faithful reverberation method would be to convolve the input signal with such a response. Direct convolution can be done by storing each sample of the impulse response as a coefficient of an FIR filter whose input is the dry signal. Direct convolution becomes easily impractical if the length of the target response exceeds small fractions of a second, as it would translate into several hundreds of taps in the filter structure. One solution is to perform the convolution block by block in the frequency domain: Given the Fourier transform of the impulse response, and the Fourier transform of a block of input signal, the two can be multiplied point by point and the result transformed back to the time domain. As this kind of processing is performed on successive blocks of the input signal, the output signal is obtained by overlapping and adding the partial results [OS89]. Thanks to the FFT computation of the discrete Fourier transform, such techniques can be significantly faster. A drawback is that, in order to be operated in real time, a block of N samples must be read and then processed while a second block is being read. Therefore, the input-output latency in samples is twice the size of a block, and this is not tolerable in practical real-time environments.

The complexity-latency trade-off is illustrated in Fig. 6.28, where the direct-form and the block-processing solutions can be located, together with a third efficient yet

low-latency solution [Gar95, Mül99]. This third realization of convolution is based on a decomposition of the impulse response into increasingly large chunks. The size of each chunk is twice the size of its predecessor, so that the latency of prior computation can be occupied by the computations related to the following impulse-response chunk. Details and discussion on convolution were presented in section 2.2.4.

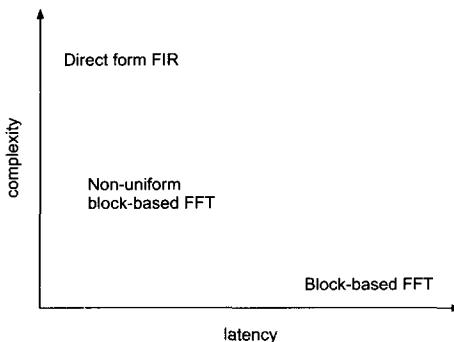


Figure 6.28 Complexity vs. latency trade-off in convolution.

Even if we have enough computer power to compute convolutions by long impulse responses in real time, there are still serious reasons to prefer reverberation algorithms based on feedback delay networks in many practical contexts. The reasons are similar to those that make a CAD description of a scene preferable to a still picture whenever several views have to be extracted or the environment has to be modified interactively. In fact, it is not easy to modify a room impulse response to reflect some of the room attributes, e.g. its high-frequency absorption, and it is even less obvious how to spatialize the echoes of the impulse response in order to get a proper sense of envelopment. If the impulse response is coming from a spatial rendering algorithm, such as ray tracing, these manipulations can be operated at the level of room description, and the coefficients of the room impulse response transmitted to the real-time convolver. In the low-latency block based implementations, we can even have faster update rates for the smaller early chunks of the impulse response, and slower update rates for the reverberant tail. However, continuous variations of the room impulse response are rendered more easily using a model of reverberation operating on a sample-by-sample basis, such as those of section 6.5.3.

Music Applications and Control

Reverberation has been used as a compositional dimension by some authors. Again, Chowning gave one of the most important examples in the piece “Turenas”, especially for the use of reverberation as a means to achieve a sense of distance, as explained in section 6.2.3.

Luigi Nono made extensive use of reverberators to extend the possibilities of actual performance halls. For instance, in [m-Non88] sudden changes in reverberation time give the impression of doors that momentarily open the view to larger spaces. Moreover, the timbral features of running reverberance, with an ultra-natural decay time of 30 s, are used to give an intense expressive character to a sustained low note of the tuba.

There are several examples of reverberation by direct convolution with the peculiar impulse response of existing spaces. In the work of some authors, such as Barry Truax or Agostino Di Scipio, reverberation is often a byproduct of granular synthesis that can be somehow controlled by governing the density and distribution of concrete or synthetic sound grains.

Feedback delay networks can be interpreted as models of 3-D resonators rather than strict reverberators. If these resonators are varied in time in their size and geometry, several kinds of interesting filtering effects can be achieved, ranging from irregular comb-filtering of the voice [m-Bat93] to colored drones carrying an intrinsic sense of depth [m-Doa98].

A new interesting perspective on spatial control of audio effects has been opened up by some recent works by Di Scipio, where the amplification chain and the listening room become parts of the composition itself as they concur to determine the overall character of a piece. For instance, in [m-DiS00] the live performance makes use of a few microphones, whose input is compared with the synthetic sound material, and the difference signals are used to control several aspects of the sound transformation process. In this way, the final result is a combination of the original material, the sound transformation algorithms, the peculiarities and variability of the room/audience combination, and the real-time actions of the performer [DiS98].

6.6 Spatial Enhancements

6.6.1 Stereo Enhancement

In this chapter, we have looked at the problem of reproducing the spatial image of a sound source using different approaches. At one extreme, one can record a couple of sound signals by means of a dummy head and reproduce it by means of headphones or transaural loudspeaker arrangements. This approach has both practical advantages and drawbacks that have previously been explained in sections 6.3.4 and 6.4.5. Moreover, when it is used to collect samples to be played by sound samplers, another severe problem occurs: the waveform loops, commonly used to lengthen a steady sound beyond its stored length, tend to magnify any spatial shift of the sound image, which is perceived as floating cyclically in space. Instead of doing such an “integral sampling”, the spatial information can be introduced in a post-processing stage by implementation of HRTF filters. However, this approach tends to consider sources as being point-like and not too close to the ears. A typical example of instrumental sound that is difficult to spatialize properly is

the piano. For the pianist, the sound source is perceived as changing its spatial centroid according to the note that is being played (the hammers are displaced along a relatively wide area) and during a single note decay (as different contributions from the soundboard, the frame, and the resonating strings become evident). It is clear that any form of binaural or transaural processing that reproduces this kind of effects would be quite complicated. Fortunately, there are simplified approaches to the rendering of spatial attributes of extended sound sources. The general idea is that of constructing a simple parametric model that captures the spatial sound features of interest. The most interesting feature is the mutual correlation between two audio channels, already defined in (6.9). If the two channels are presented binaurally, a low degree of correlation is usually perceived as an externalization of the sound image, and the fluctuations of the correlation function in time are associated with a spatial impression of the enclosed space. Some authors [Ken95b, KO83] have investigated how the cross-correlation between two audio channels is perceived through a pair of loudspeakers. The peak value Ω of the function (6.9), computed on windowed portions of discretized signal, can take values in the range between -1 and $+1$, where

$$\Omega = \begin{cases} 1 & \text{identical signals (modulo a time shift)} \\ -1 & \text{180-degrees out of phase signals (modulo a time shift)} \\ 0 & \text{uncorrelated signals.} \end{cases} \quad (6.48)$$

Experiments with noise and natural sources have shown that, in a standard stereo loudspeaker layout, changes in the degree of correlation Ω give rise to a combination of changes in apparent source width and distance, as depicted in Fig. 6.29. This picture illustrates a qualitative behavior, the actual width and distance of sound sources will depend on their own sonic nature, as well as on the kind of processing that leads to a specified value of Ω . In fact, aspects of the correlation function other than the peak value will also contribute to the perceived imagery, for instance by changing the degree of coupling between distance and width.

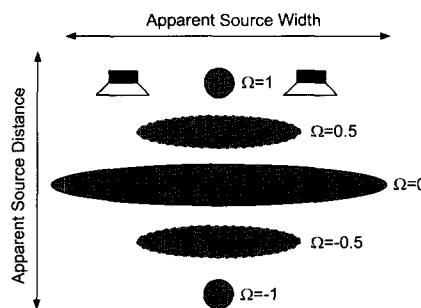


Figure 6.29 Apparent source width and distance for varying degrees of correlation Ω .

Kendall [Ken95b] proposes adjusting the degree of correlation between two audio channels by means of a couple of filters whose impulse responses are set to the desired value of Ω . Even though the perception of spatial width seems to be mainly affected

by frequencies below 2-3 kHz, a proposed filter design criterion consists of taking the full-range Inverse Fourier Transform (IFT) of two sequences having flat magnitude and random phases ϕ_1 and ϕ_2 . Values of Ω between 0 and ± 1 are obtained by composition of the two phase sequences, and taking the inverse transform with the phase sequences $(\phi_1, k\phi_1 + \phi_2)$. In this way, a catalog of FIR filters having various degrees of mutual correlation can be pre-constructed. The underlying assumption is that a flat magnitude response would not induce coloration in the decorrelated signals. However, this assumption collides with the following facts:

- As explained in section 6.4.2, a pure phase difference between two sinusoidal components at the loudspeakers induces a level difference at the ears. Therefore, the presumed magnitude transparency of the filters is not preserved at the ears.
- Between the specified frequency points, the magnitude can vary widely. One can increase the number of points (and the size of the IFT), but this comes at the expense of longer FIR filters. Besides being more expensive, if these filters extend their impulse response beyond 20 ms, a diffusion effect is introduced.

Still under the assumption that a flat magnitude response at the loudspeaker is an important requirement, an alternative decorrelation is proposed in [Ken95b], which makes use of allpass filters whose poles are randomly distributed within the unit circle. This technique ensures that the magnitude response at the loudspeakers is flat in the whole frequency range. Moreover, it is easier to implement dynamic variations of the filter coefficients without reading precomputed values in a table. Dynamic filter variations produce interesting effects that resemble variations in the geometry of a source or a room. In both the FIR and the allpass decorrelators, the filter order has to be quite high (several hundred) to achieve good degrees of decorrelation. Also, working only with random phase variations can introduce an unpleasant character, called “phasiness” [Ger92b, Ger92a], to the perceived spatial image. If we consider the Fourier transforms y_1 and y_2 of the two channels of a binaural recording, we can define the two quantities

- Apparent position: $\Re\left(\frac{y_1 - y_2}{y_1 + y_2}\right)$
- Phasiness: $\Im\left(\frac{y_1 - y_2}{y_1 + y_2}\right)$.

The phasiness is considered as a negative attribute, especially because it induces listening fatigue. In order to minimize phasiness, Gerzon [Ger92a] proposes linear phase FIR filters with irregular magnitude. Between the filters applied to the two channels, the magnitude responses should be complementary to avoid coloration. Since realizing these requirements in practice can be difficult and expensive, one possibility is to use allpass filters networked to form a multi-input multi-output allpass block.

Considering the prescriptions of Kendall, who recommends using flat-magnitude decorrelation filters, and those of Gerzon, who recommends using linear-phase filters, one might argue that in practice neither of the two ways is necessarily the

best. A third way is found by using a feedback delay network (see section 6.5.3), where two relatively uncorrelated outputs can be taken by using two distinct sets of output coefficients. The scheme for such a decorrelator is depicted in Fig. 6.30, where the d coefficients weight the contribution of the stereo input, m_l and m_r are the delays of direct signals, and m_f is the delay of the signal circulated in the FDN. The delay lengths can be set long enough to avoid coloration and coprime to each other to minimize temporal artifacts. Alternatively, if one is aware of the physical reasons for decorrelation at the ears, the delay lengths might be tuned based on some physical object involved in sound diffusion and propagation. For instance, in the case of the piano it is likely that the soundboard plays a key role in providing the appropriate decorrelation to the ears of the pianist. Therefore, the delay lengths can be tuned to match the lowest resonances of a piano soundboard.

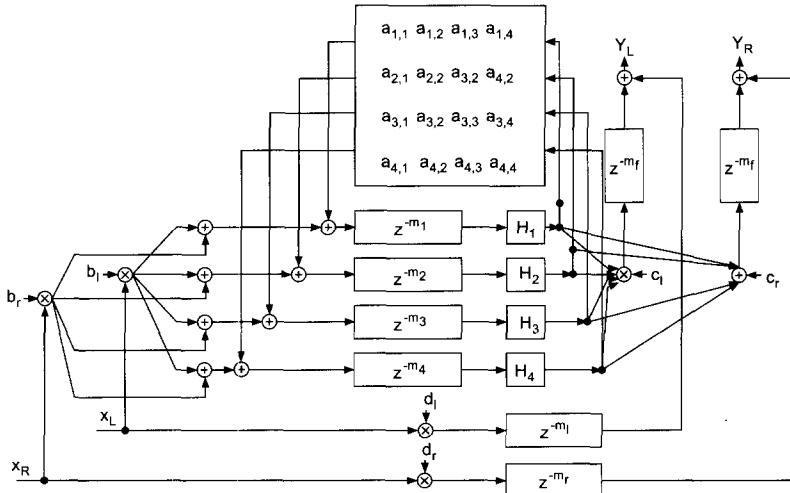


Figure 6.30 Decorrelation of a stereo input pair by means of a feedback delay network.

The model in Fig. 6.30 allows a fine tuning of the central position of the sound image by adjustment of interaural time and intensity differences, without changing the overall width of the sound image, which is essentially due to the FDN. It can also be augmented and made more accurate by means of the structural binaural model presented in section 6.3.4. Finally, the properly decorrelated signals can be played through headphones or, using the structures of section 6.4.5, through a loudspeaker pair.

Figure 6.31 shows the measured interchannel ratio of magnitudes, apparent position, and phasiness of a lowpass filtered and subsampled binaural piano recording, taken in a time window that extends for 100 ms right after the attack. The same quantities can be plotted after replacing the two binaural channels with the outputs of the network of Fig. 6.30. This kind of visualization, together with plots of the short-time inter-channel correlation, is beneficial as a complement to the aural feedback in the fine tuning stage of the decorrelator coefficients.

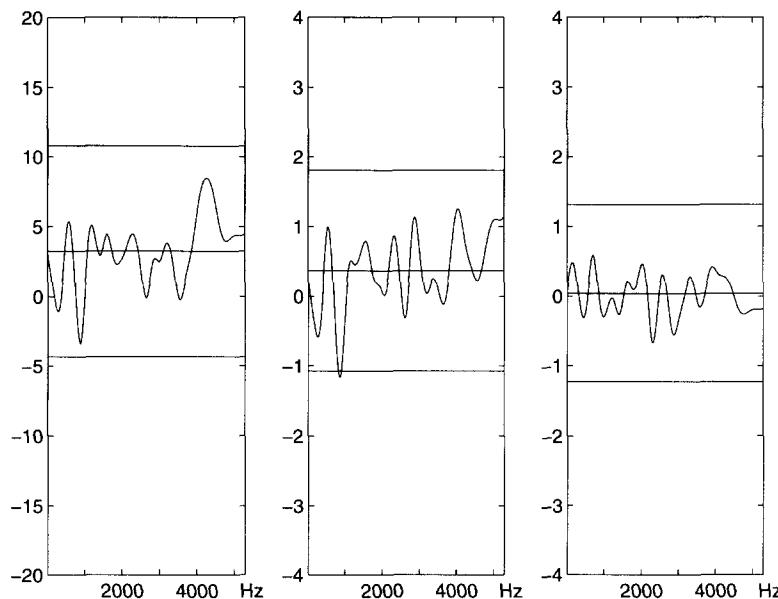


Figure 6.31 Frequency-dependent ratio of magnitudes, apparent position, and phasiness of a binaural piano sample.

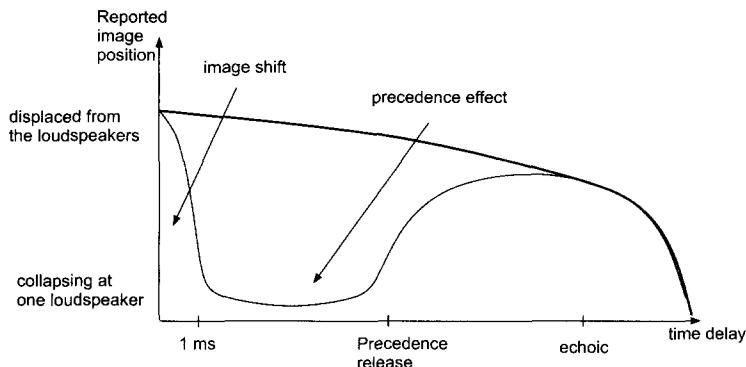


Figure 6.32 Reported apparent sound image position for perfectly correlated (thin line) and decorrelated (thick line) signals (after Kendall [Ken95b]).

As noted in [Ken95b], a decorrelator is not only useful to increase the apparent width of a sound source, but can also be effective in defeating the precedence effect to some extent. Figure 6.32 shows how the listener perceives musical sounds coming from a loudspeaker pair as a function of the relative time difference between signals (for instance, due to off-axis listening positions) and for two extreme degrees of correlation.

6.6.2 Sound Radiation Simulation

In common musical thinking, loudspeakers are often considered transparent sources of either spherical (point source) or plane (planar infinitely extended source) waves. These assumptions are also made, for the sake of simplicity, by the designers of most spatialization systems. However, no loudspeaker system is transparent because it introduces linear and nonlinear distortion, and its radiation pattern can never be described as a sphere or a plane. As a consequence, there are spatial and signal distortions that can degrade the quality of music played by means of loudspeakers. These distortions become easily audible when there are acoustic instruments amplified by loudspeaker systems. As any sound engineer can confirm, obtaining a good balance between the natural source and the loudspeakers can be quite difficult, so that delays are often introduced to deceive the audience using the precedence effect (see section 6.2.2), so that they believe that sounds are coming from the natural source only.

Sometimes, it is interesting to reproduce the radiation pattern of an acoustic source, such as a musical instrument, by means of a system of loudspeakers. This is seldom the case in regular concert layouts, but it can be effective in sound installations and sound sculptures, where the listeners are supposed to move around and through the loudspeaker system. Another special application of sound radiation simulation is found in digital pianos: even if sound samples are perfectly recorded from a high-quality instrument, the spatial impression given to the pianist by sounds as they are played through a few small loudspeakers can be quite disappointing.

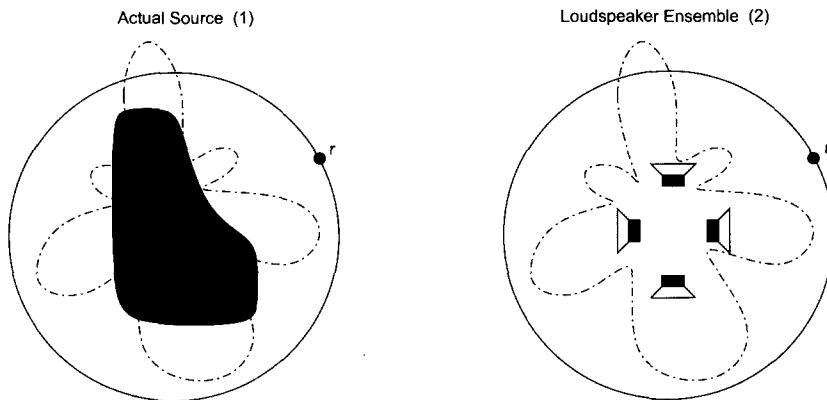


Figure 6.33 Sound source radiation measured along a boundary (1), and reconstruction of the sound field within the boundary by means of a loudspeaker set (2).

Sound radiation simulation can be formulated as a problem of reconstruction of the acoustic field in a space by means of a finite number of sources. The situation is depicted in Fig. 6.33, where the target sound source is depicted within a volume enclosed by a boundary, and the same volume encloses a set of loudspeakers. The goal is to apply some filtering to the loudspeakers so that the system on the right

has the same radiation properties as the system on the left. In mathematical terms, this is expressed by

$$\sum_{i=1}^N a_i(\omega) P_i(\mathbf{r}, \omega) = R(\mathbf{r}, \omega), \text{ for any point } \mathbf{r} \text{ on the boundary } \partial V, \quad (6.49)$$

where $P_i(\mathbf{r}, \omega)$ is the frequency response of loudspeaker i measured at point \mathbf{r} , and $R(\mathbf{r}, \omega)$ is the frequency response of the target sound source measured at the same point \mathbf{r} . If (6.49) holds at the boundary, the equivalence of the two systems of Fig. 6.33 is also true in the whole enclosed space [EH69]. The frequency responses P_i can be thought of as vectors identifying a vector space \mathcal{V} , so that the set of coefficients a_i that gives the vector closest to R is obtained by orthogonal projection of R onto \mathcal{V} (see Fig. 6.34). The projection is such that the scalar product between the distance vector $R - \sum_j a_j P_j$ and each basis vector P_j is identically zero. Since the scalar product is computed by integration over the boundary [DCW95], we have

$$(R - \sum_j a_j P_j) \cdot P_i = \oint_{\partial V} R P_i dS - \sum_j a_j \oint_{\partial V} P_j P_i dS = 0. \quad (6.50)$$

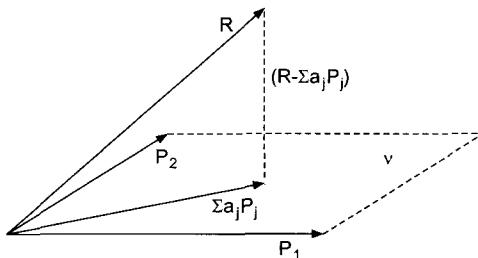


Figure 6.34 Finding the minimum distance between a vector R and a vector space \mathcal{V} by orthogonal projection.

The solution to (6.50) can be expressed as the solution to a linear system of equations in the unknowns a_j , to be solved for each frequency ω of interest. The resulting frequency responses $a_j(\omega)$, to be imposed on each audio channel, can then be approximated by filter design techniques, or inverted and implemented by direct or fast convolution. With a small number of loudspeakers, the radiation pattern can be fairly approximated only up to a few hundreds Hz [DCW95]. However, accurate tuning of low-frequency directivity can be enough to give the impression of a directional tone color similar to an acoustic source. Some researchers have started collecting directional impulse responses of musical instruments [CT98], so that a database of filter coefficients can be designed, and they can be switched and interpolated interactively in electroacoustic music performances.

6.7 Conclusion

Playing with the spatial attributes of sound has been an intriguing and challenging task for many musicians and sound designers. The multiplicity of techniques developed so far has been roughly overviewed in the previous pages. Despite the thickness of this chapter, we have certainly missed many important contributions to the field. However, we endeavored to communicate the main structural, perceptual, or technological limitations and possibilities of spatial audio. We hope that the sound designer, after reading this chapter, will be able to model some spatial features of sound or, at least, to be conscious of those features that will be part of the aesthetics of the design process rather than part of the sonic outcome.

Technological progress will stimulate more research in spatial audio in the future. A particularly promising area is that of audio embedded in everyday objects as a feasible form of display for ubiquitous computing. As the number and flexibility of sound sources are likely to increase in this new context, it is likely that new paradigms for spatial sound design will emerge.

Sound and Music

- [m-Bat93] G. Battistelli. *Frau Frankenstein*. 1993. In: CD BMG RICORDI 74321465302, 1997.
- [m-Bou84] P. Boulez. *Répons*. 1981-84. In: CD Deutsche Grammophon 457 605-2, 1998.
- [m-Cag69] J. Cage and L. Hiller. *HPSCHD*. 1967-69. In: LP Nonesuch Records H-71224, 1969. CD IDEAMA 069. Karlsruhe: ZKM, 1996.
- [m-Cho72] J. Chowning. *Turenas*. 1972. In: CD Wergo WER 2012-50, 1988.
- [m-Cif95] F. Cifariello Ciardi. *Games*. 1995. Edipan Ed., Rome. In: CD PAN 3064, 1999.
- [m-DiS00] A. Di Scipio. *5 Difference-Sensitive Circular Interactions*. 2000. In: CD of the International Computer Music Conference. Berlin, Germany, 2000. ICMA.
- [m-Doa98] R. Doati. *Inventario delle Eclissi*. 1996. In: CD annex to the book *Poetronics: Al confine tra suono, parola, tecnologia*, A. Di Vincenzo and A.G. Immertat, eds., Edizioni Tracce, Pescara – Italy, 1999.
- [m-Gua94] A. Guarnieri. *Orfeo cantando... tolse....* 1994. In: CD BMG RICORDI 74321526802, 1997.
- [m-Gua99] A. Guarnieri. *Passione secondo Matteo*. 1999. Ricordi Ed., Milan.
- [m-Non82] L. Nono. *Prometeo*. 1982. Ricordi Ed., Milan.

- [m-Non88] L. Nono. Post-Prae-Ludium per Donau. 1987. In: CD ARTIS ARCO 032, 1993.
- [m-Pis95] M. Pisati. ZONE I: zone hack a direzione virtuale. 1995. Ricordi Ed., Milan.
- [m-Sto56] K. Stockhausen. Gesang der Jünglinge. 1955–56. In: CD-3 of the Stockhausen Complete Edition, Stockhausen Verlag.
- [m-Tru85] B. Truax. Solar Ellipse. 1984–85. In: CD Wergo WER 2017-50, 1988.

Bibliography

- [AB79] J. Allen and D. Berkley. Image method for efficiently simulating small-room acoustics. *J. Acoust. Soc. Am.*, 65(4):912–915, April 1979.
- [BC00] R. Bianchini and A. Cipriani. *Virtual Sound*. ConTempo, Rome, Italy, 2000.
- [Ber92] L.L. Beranek. Concert hall acoustics - 1992. *J. Acoust. Soc. Am.*, 92(1):1–39, July 1992.
- [BD98] C.P. Brown and R.O. Duda. A structural model for binaural sound synthesis. *IEEE Trans. Speech and Audio Processing*, 6(5):476–488, Sept. 1998.
- [Bla83] J. Blauert. *Spatial Hearing: The Psychophysics of Human Sound Localization*. MIT Press, 1983.
- [BL86] J. Blauert and W. Lindemann. Auditory spaciousness: Some further psychoacoustic analyses. *J. Acoust. Soc. Am.*, 80(2):533–542, August 1986.
- [Bor84] J. Borish. Extension of the image model to arbitrary polyhedra. *J. Acoust. Soc. Am.*, 75(6):1827–1836, June 1984.
- [BV95] A. Belladonna and A. Vidolin. spAAce: un programma di spazializzazione per il Live Electronics. In *Proc. Second Int. Conf. on Acoustics and Musical Research*, pages 113–118, Ferrara, Italy, 1995.
- [CB89] D.H. Cooper and J.L. Bauck. Prospects for transaural recording. *J. Audio Eng. Soc.*, 37(1/2):3–19, Jan/Feb 1989.
- [Cho71] J.M. Chowning. The simulation of moving sound sources. *J. Audio Eng. Soc.*, 19(1):2–6, 1971. Reprinted in the *Computer Music Journal*, June 1977.

- [Cho99] J.M. Chowning. Perceptual fusion and auditory perspective. In P.R. Cook (ed), *Music, Cognition, and Computerized Sound: An Introduction to Psychoacoustics*. MIT Press, Cambridge, MA, 1999. Pages 261–275. Reprinted in the *Computer Music Journal*, June 1977.
- [CT98] P.R. Cook and D. Trueman. NBody: interactive multidirectional musical instrument body radiation simulators, and a database of measured impulse responses. In *Proc. International Computer Music Conference*, Ann Arbor, MI, pages 353–356, 1998.
- [CW93] M.F. Cohen and J.R. Wallace. *Radiosity and Realistic Image Synthesis*. Academic Press, 1993.
- [Dat97] J. Dattorro. Effect design - Part 1: Reverberator and other filters. *J. Audio Eng. Soc.*, 45(19):660–684, September 1997.
- [DCW95] P. Derogis, R. Causse, and O. Warusfel. On the reproduction of directivity patterns using multi-loudspeaker sources. In *Proc. Intern. Symp. on Musical Acoustics*, Dourdan, France, pages 387–392, 1995.
- [DiS98] A. Di Scipio. El sonido en el espacio, el espacio en el sonido. *Doce Notas Preliminares*, 2:133–157, December 1998.
- [DM98] R.O. Duda and W.L. Martens. Range dependence of the response of a spherical head model. *J. Acoust. Soc. Am.*, 104(5):3048–3058, November 1998.
- [Dur92] N. Durlach. On the externalization of auditory images. *Presence*, 1(2):251–257, Spring 1992.
- [EH69] W.C. Elmore and M.A. Heald. *Physics of Waves*. McGraw-Hill, 1969. Reprinted by Dover Publications, Inc., 1985.
- [FU98] A. Farina and E. Ugoletti. Software implementation of B-format encoding and decoding. *Preprint of the Audio Eng. Soc. Convention*, Amsterdam, Netherlands, May, 1998.
- [Gar95] W.G. Gardner. Efficient convolution without input-output delay. *J. Audio Eng. Soc.*, 43(3):127–136, March 1995.
- [Gar98a] W.G. Gardner. *3-D Audio using Loudspeakers*. Kluwer Academic Publishers, 1998.
- [Gar98b] W.G. Gardner. Reverberation algorithms. In M. Kahrs and K. Brandenburg (eds), *Applications of Digital Signal Processing to Audio and Acoustics*, Kluwer Academic Publishers, pages 85–131, 1998.
- [GB98] M.A. Gerzon and G.J. Barton. Surround sound apparatus, 1998. U.S. Patent no. 5,757,927.

- [Ger76] M.A. Gerzon. Unitary (energy preserving) multichannel networks with feedback. *Electronics Letters V*, 12(11):278–279, 1976.
- [Ger85] M.A. Gerzon. Ambisonics in multichannel broadcasting and video. *J. Audio Eng. Soc.*, 33:859–871, November 1985.
- [Ger92a] M.A. Gerzon. Optimum reproduction matrices for multispeaker stereo. *J. Audio Eng. Soc.*, 40(7/8):571–589, 1992.
- [Ger92b] M.A. Gerzon. Signal processing for simulating realistic stereo images. *Preprint of the Audio Eng. Soc. Convention*, San Francisco, October 1992.
- [GM94] W.G. Gardner and K. Martin. HRTF measurements of a KEMAR dummy-head microphone. Technical report # 280, MIT Media Lab, 1994.
- [Gri94] D. Griesinger. Subjective loudness of running reverberation in halls and stages. In *Proc. W.C. Sabine Centennial Symposium*, Cambridge, MA, pages 89–92, June 1994.
- [Gri97] D. Griesinger. The psychoacoustics of apparent source width, spaciousness and envelopment in performance spaces. *Acustica*, 83:721–731, 1997.
- [Gri99] D. Griesinger. Objective measures of spaciousness and envelopment. In *Proc. Audio Eng. Soc. Int. Conference*, Rovaniemi, Finland, pages 27–41, April 1999.
- [Hal95] H.P. Haller. *Das Experimental Studio der Heinrich-Strobel-Stiftung des Südwestfunks Freiburg 1971–1989. Die Erforschung der Elektronischen Klangumformung und ihre Geschichte*. Südwestfunk, Schriftenreiche, Rundfunkgeschichte, Band 6/1 und 6/2 Nomos Verlagsgesellschaft, Baden-Baden, Germany, 1995.
- [HW96] W.M. Hartmann and A. Wittenberg. On the externalization of sound images. *J. Acoust. Soc. Am.*, 99(6):3678–3688, 1996.
- [HZ99] J. Huopaniemi and N. Zacharov. Objective and subjective evaluation of head-related transfer function filter design. *J. Audio Eng. Soc.*, 47(4):218–239, April 1999.
- [IAS98] 3D Working Group of the Interactive Audio Special Interest Group. *Interactive 3D Audio Rendering Guidelines. Level 1.0*. MIDI Manufacturers Association, June 9, 1998.
- [IAS99] 3D Working Group of the Interactive Audio Special Interest Group. *Interactive 3D Audio Rendering Guidelines. Level 2.0*. MIDI Manufacturers Association, September 20, 1999.
- [JC91] J.-M. Jot and A. Chaigne. Digital delay networks for designing artificial reverberators. Preprint of the *Audio Eng. Soc. Convention*, Paris, February 1991.

- [Jot92] J.-M. Jot. *Etude et Réalisation d'un Spatialisateur de Sons par Modèles Physiques et Perceptifs*. PhD thesis, TELECOM, Paris 92 E 019, 1992.
- [Jot99] J.-M. Jot. Real-time spatial processing of sounds for music, multimedia, and interactive human-computer interfaces. *Multimedia Systems*, 7(1):55–69, 1999.
- [JS83] D.A. Jaffe and J.O. Smith. Extensions of the Karplus-Strong plucked string algorithm. *Computer Music J.*, 7(2):56–69, 1983.
- [Jul95] J.-P. Jullien. Structured model for the representation and the control of room acoustical quality. In *Proc. 15th Int. Conf. on Acoustics*, Trondheim, Norway, pages 517–520, 1995.
- [Kai80] T. Kailath. *Linear Systems*. Prentice-Hall, 1980.
- [KC98] A. Kulkarni and H.S. Colburn. Role of spectral detail in sound-source localization. *Nature*, 396:747–749, December 1998.
- [Ken95a] G.S. Kendall. A 3-D sound primer: directional hearing and stereo reproduction. *Computer Music J.*, 19(4):23–46, Winter 1995.
- [Ken95b] G.S. Kendall. The decorrelation of audio signals and its impact on spatial imagery. *Computer Music J.*, 19(4):71–87, Winter 1995.
- [KO83] K. Kurozumi and K. Ohgushi. The relationship between the cross-correlation coefficient of two-channel acoustic signals and sound image quality *J. Acoust. Soc. Am.*, 74:1728–1733, 1983.
- [Kuh77] G. Kuhn. Model for the interaural time differences in the azimuthal plane. *J. Acoust. Soc. Am.*, 62:157–167, July 1977.
- [Kut91] H. Kuttruff. *Room Acoustics*. Elsevier Science, Essex, 1991. 3rd ed; 1st ed 1973.
- [Kut95] H. Kuttruff. A simple iteration scheme for the computation of decay constants in enclosures with diffusely reflecting boundaries. *J. Acoust. Soc. Am.*, 98(1):288–293, July 1995.
- [KV01] M. Kubovy and D. Van Valkenburg. Auditory and visual objects. *Cognition*, 90:97–126, 2001.
- [KW92] D.J. Kistler and F.L. Wightman. A model of head-related transfer functions based on principal components analysis and minimum-phase reconstruction. *J. Acoust. Soc. Am.*, 91(3):1637–1647, March 1992.
- [Kyr98] C. Kyriakakis. Fundamental and technological limitations of immersive audio systems. *Proc. IEEE*, 86(5):941–951, May 1998.
- [LS99] C. Landone and M. Sandler. Issues in performance prediction of surround systems in sound reinforcement applications. In *Proc. DAFX-99 Digital Audio Effects Workshop*, pages 77–82, Trondheim, December 1999.

- [Mar01] W.L. Martens. Psychophysical calibration for controlling the range of a virtual sound source: multidimensional complexity in spatial auditory display. In *Int. Conf. on Auditory Display*, pages 197–207, Espoo, Finland, 2001.
- [MHV97] J. Mackenzie, J. Huopaniemi, and V. Välimäki. Low-order modeling of head-related transfer functions using balanced model truncation. *IEEE Signal Processing Letters*, 4(2):39–41, February 1997.
- [MI86] P.M. Morse and K.U. Ingard. *Theoretical Acoustics*. McGraw-Hill, 1968. Reprinted in 1986, Princeton University Press.
- [Mit98] S.K. Mitra. *Digital Signal Processing: A Computer-Based Approach*. McGraw-Hill, 2nd edition, 2001.
- [MM95] D. G. Malham and A. Myatt. 3-D sound spatialization using ambisonic techniques. *Computer Music J.*, 19(4):58–70, Winter 1995.
- [Moo79] J.A. Moorer. About this reverberation business. *Computer Music J.*, 3(2):13–18, 1979.
- [Moo82] F.R. Moore. A general model for spatial processing of sounds. *Computer Music J.*, 7(3):6–15, 1982.
- [Mor91] P.M. Morse. *Vibration and Sound*. American Institute of Physics for the Acoustical Society of America, 1991. 1st ed 1936, 2nd ed 1948.
- [Mül99] C. Müller-Tomfelde. Low-latency convolution for real-time applications. In *Proc. Audio Eng. Soc. Int. Conference*, pages 454–459, Rovaniemi, Finland, April 1999.
- [NE98] R. Nicol and M. Emerit. Reproducing 3D-sound for videoconferencing: a comparison between holophony and ambisonic. In *Proc. DAFX-98 Digital Audio Effects Workshop*, pages 17–20, Barcelona, November 1998.
- [Neu98] J.G. Neuhoff. A perceptual bias for rising tones. *Nature*, 395(6698):123–124, 1998.
- [OS89] A.V. Oppenheim and R.W. Schafer. *Discrete-Time Signal Processing*. Prentice-Hall, 1989.
- [PBJ98] J.-M. Pernaux, P. Boussard, and J.-M. Jot. Virtual sound source positioning and mixing in 5.1 implementation on the real-time system Genesis. In *Proc. DAFX-98 Digital Audio Effects Workshop*, pages 76–80, Barcelona, November 1998.
- [PKH99] V. Pulkki, M. Karjalainen, and J. Huopaniemi. Analyzing virtual sound source attributes using binaural auditory models. *J. Audio Eng. Soc.*, 47(4):203–217, April 1999.
- [Pou57] J. Poullin. Son et espace. *La Revue Musicale*, 1957.

- [Pul97] V. Pulkki. Virtual sound source positioning using vector base amplitude panning. *J. Audio Eng. Soc.*, 45(6):456–466, 1997.
- [RBM95] D. Rocchesso, O. Ballan, and L. Mozzoni. Sound spatialization for live music performance. *Proc. Second Int. Conf. on Acoustics and Musical Research*, pages 183–188, Ferrara, Italy, 1995.
- [RD01] D. Rocchesso and P. Dutilleux. Generalization of a 3-D resonator model for the simulation of spherical enclosures. *Applied Signal Processing*, 2001:15–26, 2001.
- [Roc95] D. Rocchesso. The Ball within the Box: a sound-processing metaphor. *Computer Music J.*, 19(4):47–57, Winter 1995.
- [Roc96] D. Rocchesso. *Strutture ed Algoritmi per l'Elaborazione del Suono basati su Reti di Linee di Ritardo Interconnesse*. PhD thesis, University of Padua, February 1996.
- [Roc97] D. Rocchesso. Maximally-diffusive yet efficient feedback delay networks for artificial reverberation. *IEEE Signal Processing Letters*, 4(9):252–255, September 1997.
- [RS97] D. Rocchesso and J.O. Smith. Circulant and elliptic feedback delay networks for artificial reverberation. *IEEE Transactions on Speech and Audio Processing*, 5(1):51–63, January 1997.
- [RV89] D. Rife and J. Vanderkooy. Transfer-function measurements using maximum-length sequences. *J. Audio Eng. Soc.*, 37(6):419–444, June 1989.
- [RV96] D. Rocchesso and A. Vidolin. Sintesi del movimento e dello spazio nella musica elettroacustica. In *Atti del Convegno La Terra Fertile*, L’Aquila, Italy, October 1996.
- [Sch61] M.R. Schroeder. Improved quasi-stereophony and “colorless” artificial reverberation. *J. Acoust. Soc. Am.*, 33(8):1061–1064, August 1961.
- [Sch62] M.R. Schroeder. Natural-sounding artificial reverberation. *J. Audio Eng. Soc.*, 10(3):219–233, July 1962.
- [Sch70] M.R. Schroeder. Digital simulation of sound transmission in reverberant spaces. *J. Acoust. Soc. Am.*, 47(2):424–431, 1970.
- [Sch73] M.R. Schroeder. Computer models for concert hall acoustics. *American Journal of Physics*, 41:461–471, 1973.
- [SF85] J.O. Smith and B. Friedlander. Adaptive interpolated time-delay estimation. *IEEE Trans. Aerospace and Electronic Systems*, 21(2):180–199, March 1985.

- [SL61] M.R. Schroeder and B. Logan. “Colorless” artificial reverberation. *J. Audio Eng. Soc.*, 9:192–197, July 1961. Reprinted in the *IRE Trans. on Audio*.
- [Smi85] J.O. Smith. A new approach to digital reverberation using closed waveguide networks. In *Proc. International Computer Music Conference*, pages 47–53, Vancouver, Canada, 1985. Also available in [Smi87].
- [Smi86] J.O. Smith. Elimination of limit cycles and overflow oscillations in time-varying lattice and ladder digital filters. In *Proc. IEEE Conf. Circuits and Systems*, San Jose, California, May 1986. Longer version also available in [Smi87].
- [Smi87] J.O. Smith. Music applications of digital waveguides. Report stan-m-39, CCRMA - Stanford University, Stanford, California, 1987.
- [Smi92] J.O. Smith. Physical modeling using digital waveguides. *Computer Music J.*, 16(4):74–91, Winter 1992.
- [SP82] J. Stautner and M. Puckette. Designing multichannel reverberators. *Computer Music J.*, 6(1):52–65, Spring 1982.
- [Vai93] P.P. Vaidyanathan. *Multirate Systems and Filter Banks*. Prentice-Hall, 1993.
- [ZGM92] B. Zhou, D. Green, and J. Middlebrooks. Characterization of external ear impulse responses using golay codes. *J. Acoust. Soc. Am.*, 92:1169–1171, 1992.
- [ZF90] E. Zwicker and H. Fastl. *Psychoacoustics: Facts and Models*. Springer-Verlag, Berlin, Germany, 1990.

Chapter 7

Time-segment Processing

P. Dutilleux, G. De Poli, U. Zölzer

7.1 Introduction

In this chapter we discuss several time domain algorithms which are a combination of smaller processing blocks like amplitude/phase modulators, filters and delay lines. These effects mainly influence the pitch and the time duration of the audio signal. We will first introduce some basic effects like variable speed replay and pitch-controlled resampling. They are all based on delay line modulation and amplitude modulation. Then we will discuss two approaches for time stretching (time scaling) of audio signals. They are based on an analysis stage, where the input signal is divided into segments (blocks) of fixed or variable length, and a synthesis stage where the blocks of the analysis stage are recombined by an overlap and add procedure. These time stretching techniques perform time scaling without modifying the pitch of the signal. The fourth section focuses on pitch shifting, and introduces three techniques: block processing based on time stretching and resampling, delay line modulation and pitch-synchronous block processing. Block processing based on delay line modulation performs pitch shifting by scaling the spectral envelope of each block. Pitch-synchronous block processing performs pitch shifting by resampling the spectral envelope of each block and thus preserving the spectral envelope. The last section on time shuffling and granulation presents a more creative use of time-segment processing. Short segments of the input signal are freely assembled and time placed in the output signal. In this case the input sound can be much less recognizable. The wide choice of strategies for segment organization implies a sound composition attitude from the user.

7.2 Variable Speed Replay

Introduction

Analog audio tape recorders allow replay mode over a wide range of tape speeds. Particularly in fast forward/backward transfer mode, a monitoring of the audio signal is possible which is used to locate a sound. During faster playback the pitch of the sound is raised and during slower playback the pitch is lowered. With this technique the duration of the sound is lengthened, if the tape is slowed down, and shortened if the tape speed is increased. Figure 7.1 illustrates a sound segment which is lengthened and shortened and their corresponding spectra.

Signal Processing

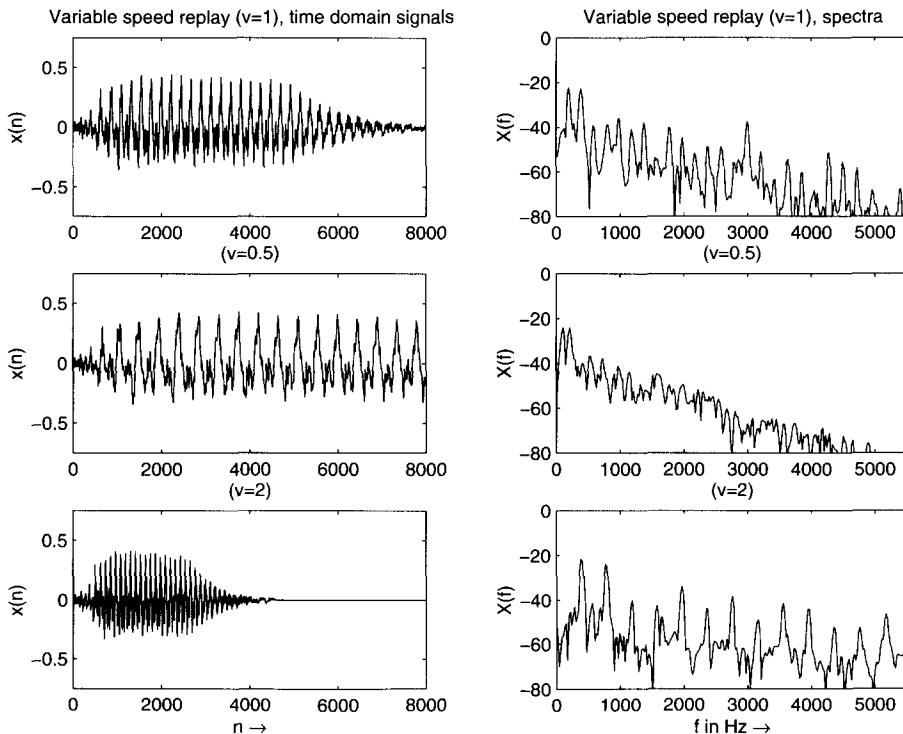


Figure 7.1 Pitch shifting: Variable speed replay leads to time compression/expansion and compression and expansion of the spectral envelope.

The phrase “variable speed replay” is used to mean that what has happened initially during the time period $nT_{s,in}$ is now happening during

$$nT_{s,replay} = nT_{s,in}/v \quad (7.1)$$

at the relative speed v , where $T_{s,in}$ and $T_{s,replay}$ are the initial and replay sampling periods. Time expansion corresponds to $v < 1$. A straightforward method of implementing the variable speed replay is hence to modify the sampling frequency while playing back the sound according to

$$f_{s,replay} = f_{s,in} \cdot v \quad (7.2)$$

where $f_{s,in}$ and $f_{s,replay}$ are the initial and replay sampling frequencies. One should distinguish whether the output should be digital or may be analog. If the output is analog, then a very effective method is to modify the sampling frequency of the output DAC. The spectrum of the signal is scaled by v and the analog reconstruction filter should be tuned in order to remove the spectral images after the conversion [Gas87, Mas98].

If a digital output is required, then a sampling frequency conversion has to be performed between the desired replay frequency $f_{s,replay}$ and the output sampling frequency $f_{s,out}$ which is usually equal to $f_{s,in}$.

If $v < 1$ (time expansion) then $f_{s,in} > f_{s,replay} < f_{s,out}$ and more output samples are needed than available from the input signal. The output signal is an interpolated (over-sampled) version by the factor $1/v$ of the input signal. If $v > 1$ (time compression) then $f_{s,in} < f_{s,replay} > f_{s,out}$ and less output samples than available in the input signal are necessary. The input signal is decimated by the factor v . Before decimation, the bandwidth of the input signal has to be reduced to $f_{s,replay}/2$ by a digital lowpass filter [McN84]. The quality of the sampling rate conversion depends very much on the interpolation filter used. A very popular method is the linear interpolation between two adjacent samples. A review of interpolation methods can be found in [Mas98, CR83].

A discrete-time implementation can be achieved by increasing/decreasing the transfer rate of a recorded digital audio signal to the DA converter, thus changing the output sampling frequency compared to the recording sampling frequency. If the output signal has to be in digital format again, we have to resample the varispeed analog signal with the corresponding sampling frequency. A discrete-time implementation without a DA conversion and new AD conversion was proposed in [McN84] and is shown in Fig. 7.2. It makes use of multirate signal processing techniques and performs an approximation of the DA/AD conversion approach. A further signal processing algorithm to achieve the acoustical result of a variable speed replay is the delay line modulation with a constant pitch change, which will be discussed in section 7.4.3.

Musical Applications and Control

As well as for analog tape-based audio editing, the variable speed replay is very popular in digital audio editing systems. See [m-Wis94c, ID 2.9 and 2.10] for a straightforward demonstration of the effect on a voice signal.

The effect of tape speed transposition has been used by Les Paul in the piece called “Whispering” in 1951 [Lee72]. This method is very often used in electro-acoustic music when the pitch of concrete sounds cannot be controlled at the time

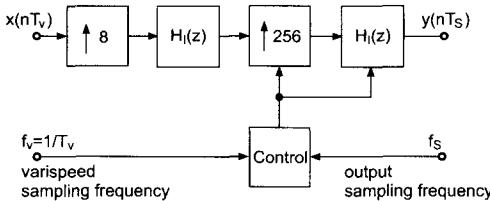


Figure 7.2 Variable speed replay scheme.

of recording. P. Schaeffer designed the *Phonogène chromatique* to transpose a sound to any one of the 12 degrees of the equal tempered scale. The device was based on a tape recorder with 12 capstans and pinch rollers. The operation of the pinch rollers could be controlled by a piano-like keyboard. An additional gear extended the range of operation to two octaves [Mol60, p. 71];[Roa96, p. 119];[Ges00]. Jacques Poullin developed another version, the *Phonogène à coulisse*, which allowed continuous speed modifications. A pair of cones, with a friction wheel in between, constitutes a variable-ratio mechanical link between the motor and the capstan of the tape player. The position of the friction wheel, and hence the replay speed, is controlled by a mechanical lever. Stockhausen, in “Hymnen”, transposed orchestral sounds to give them an overwhelming and apocalyptic character [Chi82, p. 53].

In computer music too, the variable speed replay provides an effective transposition scheme. J.-C. Risset says: by “mixing a sound with transpositions of itself with a minute frequency difference (say, a twentieth of a Hertz), one can turn steady periodic tones into a pattern where the harmonics of the tone wax and wane at different rates, proportional to the rank of the harmonic” [Ris98, p. 255];[m-INA3, Sud]. In “The Gates of H.”, Ludger Brümmer exploits the fact that variable speed replay modifies both the pitch and the duration of a sample [m-Bru93, 14’40”-17’25”]. Seven copies of the same phrase, played simultaneously at speeds 7.56, 4.49, 2.24, 1.41, 0.94, 0.67, 0.42, 0.31 are overlapped. The resulting sound begins with a complex structure and an extended spectrum. As the piece continues, the faster copies vanish and the slower versions emerge one after the other. The sound structure simplifies and it evolves towards the very low registers.

The character of the transposed sounds is modified because all the features of the spectrum are simultaneously scaled. The formants are scaled up leading to a “Mickey Mouse effect” or down, as if the sounds were produced by oversized objects. The time structure is modified as well. The transients are spread or contracted. A vibrato in the initial sound will lose its character and will appear as a slower or faster modulation. The sounds can also be played at negative speeds. A speed -1 yields a sound with the same average spectrum although sounding very different. Think about speech or percussive sounds played backwards. Other transposition schemes that are free from these drawbacks are achieved by more sophisticated methods described in further sections of this book.

A particular application was desired by the composer Kiyoshi Furukawa. He wanted a sampler for which the speed would be controlled by the amplitude of an

acoustical instrument. A sound is stored in a sampler and is played as a loop. In the meantime, the RMS amplitude of an incoming controlling signal is computed and time averaged with independent attack and decay time constants. This amplitude is converted to decibels and scaled before driving the speed of the sampler. The parameters have to be tuned in such a way that the speed remains within a valid range and the speed variations are intimately related to the loudness and the dynamics of the instrument (see Fig. 7.3).

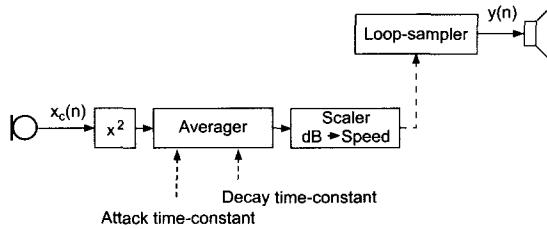


Figure 7.3 A loop-sampler controlled by an acoustical signal.

This effect is controlled by a clarinet in “Swim, swan” and by a viola in “den ungeborenen Göttern” [m-Fur93, m-Fur97]. The pitch of the acoustical instrument selects words out of a predefined set whereas the loudness controls the replay speed of these words.

7.3 Time Stretching

Introduction

In order to understand the issue of time stretching, let us take the example of a signal whose duration does not fit the time slot that is allocated to its application. Think about a speaker that has already recorded 33 seconds of speech but whose contribution to a commercial may not be longer than 30 seconds. If he does not want to record his text again, the sound engineer may artificially contract his speech by 10%. With the term “time stretching” we mean the contraction or expansion of the duration of an audio signal (time compression, time expansion, time scaling → signal processing term). We have studied in 7.2 a method that alters the duration of a sound, the variable speed replay, but it has the drawback of simultaneously transposing the sound. The *Harmonizer* could be used to transpose the sound in the opposite direction and the combination of both methods leads to a time stretching algorithm.

The main task of time stretching algorithms is to shorten or lengthen a sound file of M samples to a new particular length $M' = \alpha \cdot M$, where α is the scaling factor. For performing time stretching algorithms the sound file has to be available in a stored form on a storage medium like a sampler, DAT or a hard disc. Time stretching of a sequence of audio samples is demonstrated in Fig. 7.4. The original signal is shown in the upper plot. The middle plot shows the sequence which is

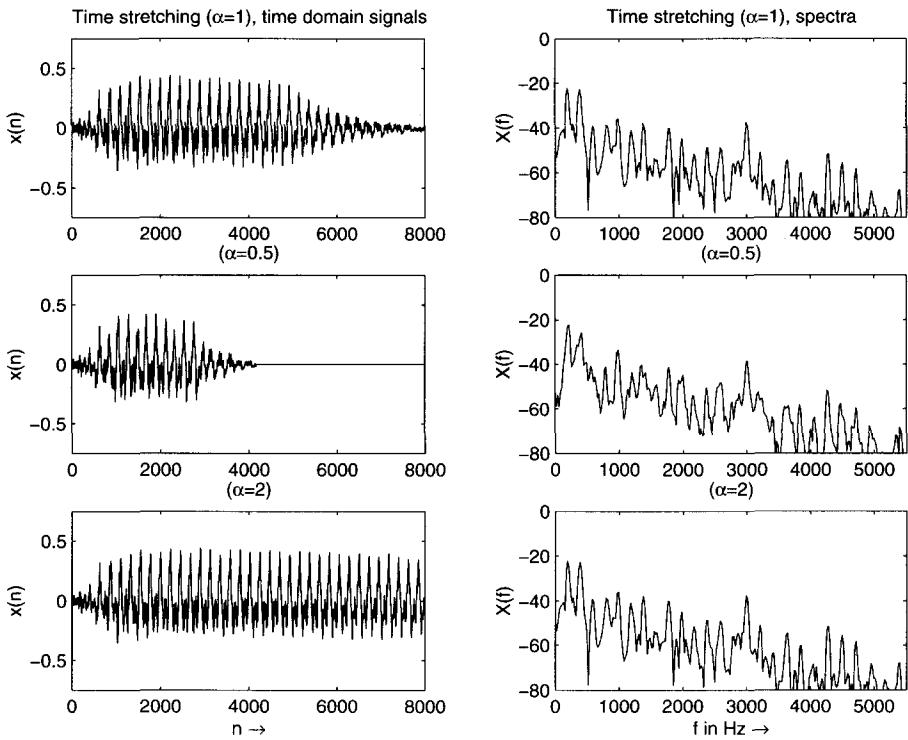


Figure 7.4 Time stretching with scaling factor $\alpha = 0.5, 2$.

shortened by a scaling factor $\alpha = 0.5$ and the lower plot shows the stretching by a scaling factor $\alpha = 2$.

Signal Processing

The intended time scaling does not correspond to the mathematical time scaling as realized by vary-speed. We rather require a scaling of the perceived timing attributes, such as speaking rate, without affecting the perceived frequency attributes, such as pitch. We could say that we want the time scaled version of an acoustic signal to be perceived as the same sequence of acoustic events as the original signal being reproduced according to a scaled time pattern. The time stretching algorithms should not affect the pitch or the frequency contents of the processed signals. This is demonstrated by the corresponding spectra (first 2000 samples) of the discrete-time signals in Fig. 7.4. For comparison only the traditional technique for time stretching based on the variable speed replay introduces a pitch shift (see section 7.2 and Fig. 7.1). The basic idea of time stretching by time-segment processing is to divide the input sound into segments. Then if the sound is to be lengthened, some segments are repeated, while if the sound is to be shortened, some segments are discarded. A possible problem is amplitude and phase discontinuity at the boundaries

of the segments. Amplitude discontinuities are avoided by partially overlapping the blocks, while phase discontinuities are avoided by a proper time alignment of the blocks. Two different strategies will be presented in subsections 7.3.2 and 7.3.3.

Applications

Special machines such as the *Phonogène universel* of Pierre Schaeffer or the Tempophon used by Herbert Eimerts allowed alteration of the time duration as well as the pitch of sounds. The *Phonogène* found many applications in *musique concrète* as a “time regulator”. In its composition “Epitaph für Aikichi Kuboyama”, Herbert Eimerts uses the Tempophon in order to iterate spoken word fragments. The device allowed the scanning of syllables, vowels and plosives and could make them shorter, longer or iterate them at will [Hal95, p. 13];[m-Eim62].

As mentioned in the Introduction, the stretching of signals can be used to match their duration to an assigned time-slot. In Techno music, different pieces of music are played one after the other as a continuous stream. This stream is supposed to have only very smooth tempo or *bpm* (beat per minute) transitions although the musical excerpts usually do not have the same tempo. In order to adjust the tempo to each other, the disc jockey modifies the replay speeds at the transition from one excerpt to the other. This method leads to temporary pitch modifications which could be objectionable. The use of time stretching methods could eliminate this problem.

After a brief presentation of the technology of the Phonogène, the following sections discuss two signal processing techniques which perform time stretching without pitch modifications.

7.3.1 Historical Methods - Phonogène

Fairbanks, Everitt and Jaeger report in 1954 on a modified tape recorder for time or frequency compression-expansion of speech [Lee72, Lar98]. Springer develops a similar machine [Spr55, Spr59] and Pierre Schaeffer praises a machine called *Phonogène universel* that was designed as a combination of the aforementioned *Phonogène chromatique* and *Phonogène à coulisse* with the rotating head drum of Springer [Mol60, p. 71-76];[Sch73, p. 47-48]; [m-Sch98, CD2, ID. 50-52];[Pou54, PS57, Ges00].

The modified tape recorder has several playback heads mounted on a rotating head drum. The absolute speed of the tape at the capstan determines the duration of the sound whereas the relative speed of the heads to that of the tape determines the amount of transposition. By electrical summation of the outputs of the different heads, a continuous sound is delivered (Fig. 7.5). Moles reports a typical operating range of +10% to -40% [Mol60, p. 76]. The Springer machine was also known as *Laufzeitregler* or *Zeitdehner* [Car92, p. 479-480];[End97].

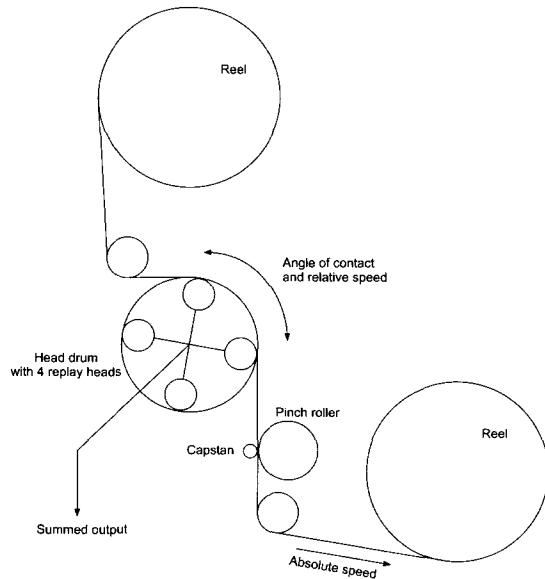


Figure 7.5 Tape-based time compression-expansion system (After [Mol60]).

7.3.2 Synchronous Overlap and Add (SOLA)

A simple algorithm for time stretching based on correlation techniques is proposed in [RW85, MEJ86]. The input signal is divided into overlapping blocks of a fixed length, as shown in Fig. 7.6. In a second step these overlapping blocks are shifted according to the time scaling factor α . Then the similarities in the area of the overlap intervals are searched for a discrete-time lag of maximum similarity. At this point of maximum similarity the overlapping blocks are weighted by a fade-in and fade-out function and summed sample by sample.

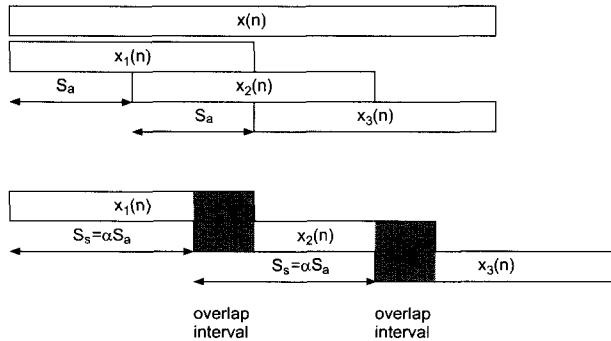


Figure 7.6 SOLA time stretching.

Algorithm description:

1. Segmentation of the input signal into blocks of length N with time shift of S_a samples.
2. Repositioning of blocks with time shift $S_s = \alpha \cdot S_a$ with scaling factor α .
3. Computation of the cross-correlation

$$r_{x_{L1}x_{L2}}(m) = \frac{1}{L} \sum_{n=0}^{L-m-1} x_{L1}(n) \cdot x_{L2}(n+m), \quad 0 \leq m \leq L \quad (7.3)$$

between $x_{L1}(n)$ and $x_{L2}(n)$, which are the segments of $x_1(n)$ and $x_2(n)$ in the overlap interval of length L .

4. Extracting the discrete-time lag k_m where the cross-correlation $r_{x_{L1}x_{L2}}(k_m) = r_{\max}$ has its maximum value (see Fig. 7.7a).
5. Using this discrete-time lag k_m , *fade-out* $x_1(n)$ and *fade-in* $x_2(n)$.
6. Overlap-add of $x_1(n)$ and $x_2(n)$ for new output signal.

Figure 7.7 illustrates the difference between simple overlap and add with fade-out and fade-in of the blocks and the refined synchronous method with the point of maximum similarity k_m . The SOLA implementation leads to time scaling with small complexity, where the parameters S_a , N , L are independent of the pitch period of the input signal.

The following M-file 7.1 demonstrates the implementation of the SOLA time scaling algorithm:

```
M-file 7.1 (TimeScaleSOLA.m)
% TimeScaleSOLA.m
% Time Scaling with Synchronized Overlap and Add
%
% Parameters:
%
% analysis hop size      Sa = 256 (default parameter)
% block length           N  = 2048 (default parameter)
% time scaling factor    0.25 <= alpha <= 2
% overlap interval        L  = 256*alpha/2

clear all,close all

[signal,Fs] = wavread('x1.wav');
DAFx_in = signal';

Sa = input('Analysis hop size Sa in samples = ');
N = input('Analysis block size N in samples = ');
```

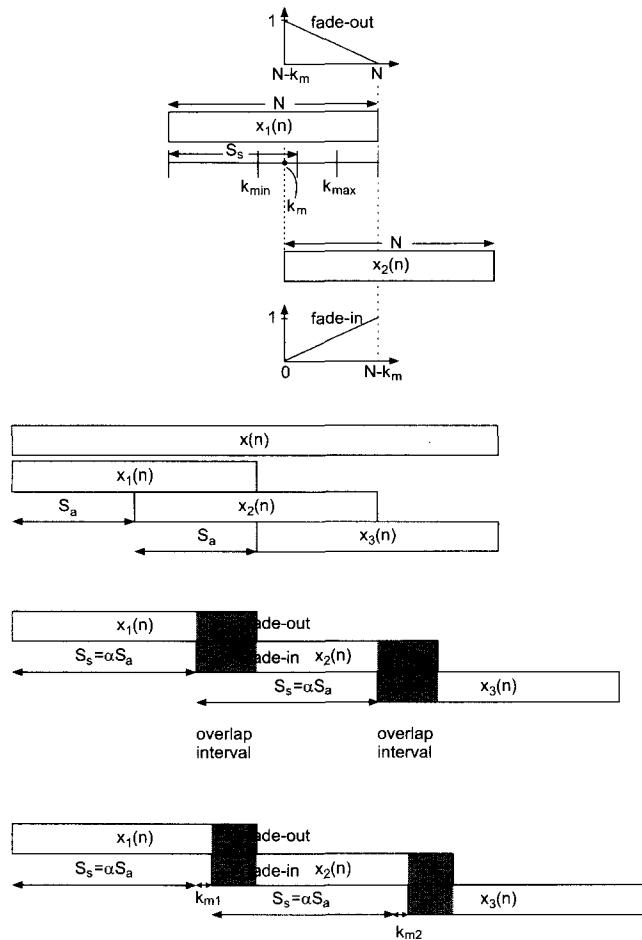


Figure 7.7 SOLA: cross-correlation and time stretching.

```

if Sa > N
    disp('Sa must be less than N !!!')
end
M = ceil(length(DAFx_in)/Sa);

% Segmentation into blocks of length N every Sa samples
% leads to M segments

alpha = input('Time stretching factor alpha      = ');
Ss  = round(Sa*alpha);
L   = input('Overlap in samples (even)      = ');

```

```

if Ss >= N disp('alpha is not correct, Ss is >= N')
elseif Ss > N-L disp('alpha is not correct, Ss is > N-L')
end

DAFx_in(M*Sa+N)=0;
Overlap = DAFx_in(1:N);

% **** Main TimeScaleSOLA loop ****
for ni=1:M-1
    grain=DAFx_in(ni*Sa+1:N+ni*Sa);
    XCORRsegment=xcorr(grain(1:L),Overlap(1,ni*Ss:ni*Ss+(L-1)));
    [xmax(1,ni),index(1,ni)]=max(XCORRsegment);
    fadeout=1:(-1/(length(Overlap)-(ni*Ss-(L-1)+index(1,ni)-1))):0;
    fadein=0:(1/(length(Overlap)-(ni*Ss-(L-1)+index(1,ni)-1))):1;
    Tail=Overlap(1,(ni*Ss-(L-1))+ ...
                  index(1,ni)-1:length(Overlap)).*fadeout;
    Begin=grain(1:length(fadein)).*fadein;
    Add=Tail+Begin;
    Overlap=[Overlap(1,1:ni*Ss-L+index(1,ni)-1) ...
              Add grain(length(fadein)+1:N)];
end;
% **** end TimeScaleSOLA loop ****
% Output in WAV file
sound(Overlap,44100);
wavwrite(Overlap,Fs,'x1_time_stretch');

```

7.3.3 Pitch-synchronous Overlap and Add (PSOLA)

A variation of the SOLA algorithm for time stretching is the Pitch Synchronous Overlap and Add (PSOLA) algorithm proposed by Moulines *et al.* [HMC89, MC90] especially for voice processing. It is based on the hypothesis that the input sound is characterized by a pitch, as for example human voice and monophonic musical instruments.

In this case PSOLA can exploit the knowledge of the pitch to correctly synchronize the time segments, avoiding pitch discontinuities. When we perform time stretching of an input sound, the time variation of the pitch $P(t)$ should be stretched accordingly. If $\tilde{t} = \alpha t$ describes the time scaling function or time warping function that maps the time t of the input signal into the time \tilde{t} of the output signal, the local pitch of the output signal $\tilde{P}(\tilde{t})$ will be defined by $\tilde{P}(\tilde{t}) = \tilde{P}(\alpha t) = P(t)$. More generally, when the scaling factor α is not constant, a nonlinear time scaling function can be defined as $\tilde{t} = \mathcal{T}(t) = \int_0^t \alpha(\tau)d\tau$ and used instead of $\tilde{t} = \alpha t$.

The algorithm is composed of two phases: the first phase analyses and segments the input sound (see Fig. 7.8), and the second phase synthesizes a time stretched version by overlapping and adding time segments extracted by the analysis algorithm.

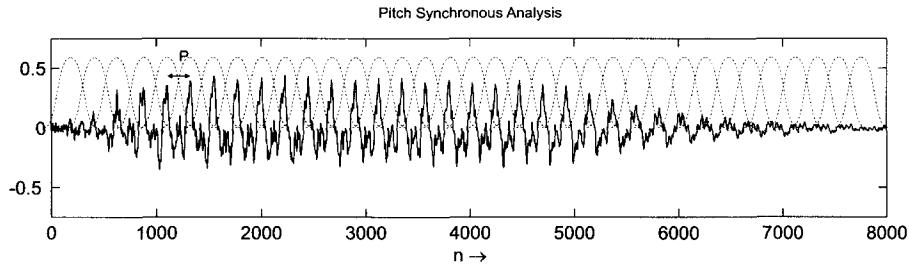


Figure 7.8 PSOLA: Pitch analysis and block windows.

Analysis algorithm (see Fig. 7.9):

1. Determination of the pitch period $P(t)$ of the input signal and of time instants (pitch marks) t_i . These pitch marks are in correspondence with the maximum amplitude or glottal pulses at a pitch synchronous rate during the periodic part of the sound and at a constant rate during the unvoiced portions. In practice $P(t)$ is considered constant $P(t) = P(t_i) = t_{i+1} - t_i$ on the time interval (t_i, t_{i+1}) .
2. Extraction of a segment centered at every pitch mark t_i by using a Hanning window with the length $L_i = 2P(t_i)$ (two pitch periods) to ensure fade-in and fade-out.

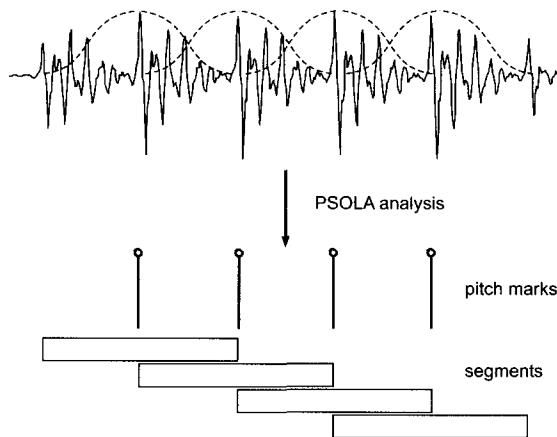


Figure 7.9 PSOLA pitch analysis.

Synthesis algorithm (see Fig. 7.10): for every synthesis pitch mark \tilde{t}_k

1. Choice of the corresponding analysis segment i (identified by the time mark t_i) minimizing the time distance $|\alpha t_i - \tilde{t}_k|$.

2. Overlap and add the selected segment. Notice that some input segments will be repeated for $\alpha > 1$ (time expansion) or discarded when $\alpha < 1$ (time compression).
3. Determination of the time instant \tilde{t}_{k+1} where the next synthesis segment will be centered, in order to preserve the local pitch, by the relation

$$\tilde{t}_{k+1} = \tilde{t}_k + \tilde{P}(\tilde{t}_k) = \tilde{t}_k + P(t_i).$$

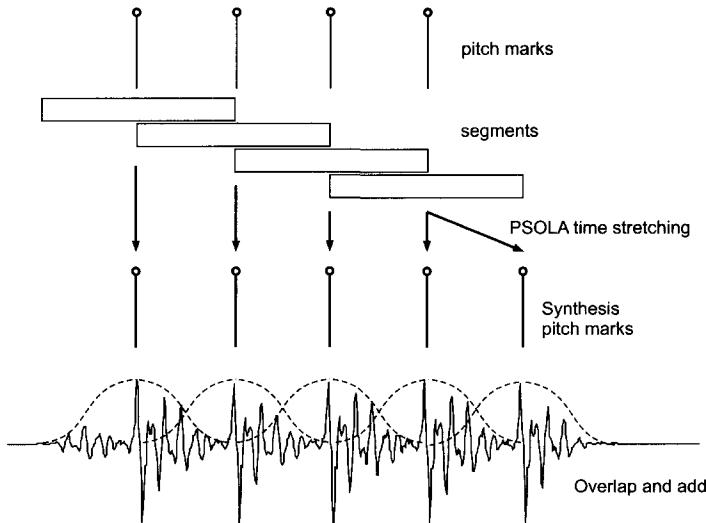


Figure 7.10 PSOLA synthesis for time stretching.

The basic PSOLA synthesis algorithm can be implemented in MATLAB by the following M-file 7.2:

```
M-file 7.2 (psola.m)
function out=psola(in,m,alpha,beta)
%     in      input signal
%     m      pitch marks
%     alpha  time stretching factor
%     beta   pitch shifting factor

P = diff(m);    %compute pitch periods

if m(1)<=P(1), %remove first pitch mark
    m=m(2:length(m));
    P=P(2:length(P));
end
```

```

if m(length(m))+P(length(P))>length(in) %remove last pitch mark
    m=m(1:length(m)-1);
else
    P=[P P(length(P))];
end

Lout=ceil(length(in)*alpha);
out=zeros(1,Lout); %output signal

tk = P(1)+1;          %output pitch mark

while round(tk)<Lout
    [minimum i] = min( abs(alpha*m - tk) ); %find analysis segment
    pit=P(i);
    gr = in(m(i)-pit:m(i)+pit) .* hanning(2*pit+1);
    iniGr=round(tk)-pit;
    endGr=round(tk)+pit;
    if endGr>Lout, break; end
    out(iniGr:endGr) = out(iniGr:endGr)+gr; %overlap new segment
    tk=tk+pit/beta;
end %while

```

It should be noticed that the determination of the pitch and of the position of pitch marks is not a trivial problem and could be difficult to implement robustly in real-time. Stretching factors typically range from $\alpha = 0.25$ to 2 for speech. Audible buzziness appears in unvoiced sound when larger values are applied, due to the regular repetition of identical input segments. In order to prevent the algorithm from introducing such an artificial short-term correlation in the synthesis signal, it is advisable to reverse the time axis of every repeated version of an unvoiced segment. With such an artifice, speech can be slowed down by a factor of four, even though some tonal effect is encountered in voiced fricatives, which combine voiced and unvoiced frequency regions and thus cannot be reversed in time.

It is possible to further exploit the analysis phase. In fact, uniformly applied time stretching can produce some artifacts on the non-periodic parts of the sound. For example a plosive consonant can be repeated if the synthesis algorithm chooses the time segment containing the consonant twice. The analysis can then be extended in order to detect the presence of fast transitions. During synthesis, the time scale will not be modified at these points, thus the segments will not be repeated. This approach can be generalized for non-speech sounds where a large time scale change during transitions (e.g. attacks) would dramatically change the timbre identity. Also in this case it is possible to limit time stretching during transitions and apply it mainly to the steady state portion of the input sound. This technique is usually applied to digital musical instruments based on wavetable synthesis. On the other hand, the deformation of transient parts can be considered an interesting timbre transformation and can be appreciated as a musically creative audio effect.

7.4 Pitch Shifting

Introduction

Transposition is one of the basic tools of musicians. When we think about providing this effect by signal processing means, we need to think about the various aspects of it. For a musician, transposing means repeating a melody after pitch shifting it by a fixed interval. Each time the performer transposes the melody, he makes use of a different register of his instrument. By doing so, not only the pitch of the sound is modified but also the timbre is affected.

In the realm of DAFX, it is a matter of choice to transpose without taking into account the timbre modification or whether the characteristic timbre of the instrument has to be maintained in each of its registers. The first method could be called “variable timbre transposition” whereas the second approach would be called “constant timbre transposition”. To get an insight into the problem we have to consider the physical origins of the audio signal.

The timbre of a sound heavily depends on the organization of its spectrum. A model can be derived from the study of the singing voice. The pitch of a singing voice is determined by the vocal chords and it can be correlated with the set of frequencies available in the spectrum. The timbre of the voice is mainly determined by the vocal cavities. Their effect is to emphasize some parts of the spectrum which are called formants. A signal model can be derived where an excitation part is modified by a resonance part. In the case of the voice, the excitation is provided by the vocal chords, hence related to the frequencies of the spectrum, whereas the resonances correspond to the formants. When a singer transposes a tune, he has, to some extent, the possibility of modifying the pitch and the formants independently. In a careful signal processing implementation of this effect, each of these two aspects should be considered.

If only the spectrum of the excitation is stretched or contracted, a pitch transposition up or down, with a constant timbre, is achieved. If only the resonances are stretched or contracted, then the pitch remains the same but the timbre is varied. The harmonic singing relies on this effect. If both excitation and resonance are deliberately and independently altered, then we enter the domain of effects that can be perceived as unnatural, but that might have a vast musical potential.

The separation of a sound into its excitation and resonance part is a complex process that will be addressed in Chapter 9. We will present here methods which simultaneously alter both aspects such as the harmonizer or pitch shifting by delay-line modulation in section 7.4.3. A more refined method based on PSOLA, which allows pitch shifting with formant preservation, will be discussed in section 7.4.4. For more advanced pitch shifting methods we refer to Chapters 8-11.

Musical Applications

Typical applications of pitch shifting in pop music are the correction of the intonation of instruments or singers as well as the production of an effect similar to a

chorus. When the voice of a singer is mixed with copies of itself that are slightly transposed, a subtle effect appears that gives the impression that one is listening to a choir instead of a single singer.

The harmonizer can also produce surprising effects such as a man speaking with a tiny high pitched voice or a female with a gritty low-pitched one. Extreme sounds can be produced such as the deep snare drum sound on David Bowie's "Let's Dance" record [Whi99]. It has also been used for scrambling and unscrambling speech [GRH73]. In combination with a delay line and with feedback of the transposed sound to the input, a kind of spiral can be produced where the sound is always transposed higher or lower at each iteration.

A subtle effect, similar to a phasing, can be achieved with a set of harmonizers [Dut88] coupled in parallel and mixed to the input sound, as shown in Fig. 7.11. The transposition ratio of the n^{th} harmonizer should be set to $1 + nr$ where r is of the order of $1/3000$. If f_0 is the pitch of the sound, the outputs of the n^{th} harmonizer will provide a pitch of $f_0 + n\Delta f$, where $\Delta f = rf_0$. If Δf is small enough (a few $1/100$ Hz) the interferences between the various outputs of the harmonizers will be clearly audible. When applied, for example, to a low-pitched tuba sound, one harmonic after the other will be emphasized. Flanging and chorus effects can also be achieved by setting the pitch control for a very slight amount of transposition (say, $1/10$ to $1/5$ of a semitone) and adding regeneration [And95, p. 53]. It appears here that tuning an audio effect is very dependent on the sound being processed. It frequently happens that the tuning has to be adjusted for each new sound or each new pitch.

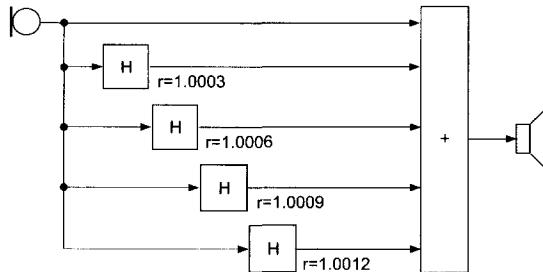


Figure 7.11 A set of harmonizers that produce a phasing-like effect. It is particularly effective for low-pitched (typ. 100 Hz) signals of long duration.

Hans Peter Haller describes in [Hal95, pp. 51-55] some applications of the harmonizer for the production of musical works from Luigi Nono and André Richard.

7.4.1 Historical Methods - Harmonizer

The tape-based machines described in 7.3.1 were also able to modify the pitch of sounds while keeping their initial duration. The *Phonogène universel* was bulky and could not find a broad diffusion but in the middle of the 1970s, a digital device appeared that was called a *Harmonizer*. It implemented in the digital domain a

process similar to that of the *Phonogène universel*. From there on the effect became very popular. Since *Harmonizer* is a trade mark of the Eventide company, other companies offer similar devices under names such as *pitch transposer* or *pitch shifter*.

The main limitation of the use of the harmonizer is the characteristic quality that it gives to the processed sounds. Moles states that the operating range of the *Phonogène universel*, used as a pitch regulator, was at least -4 to $+3$ semitones [Mol60, p. 74]. Geslin estimates that the machines available in the late sixties found application in *musique concrète* also at much larger transposition ratios [Ges00].

The digital implementations in the form of the harmonizer might allow for a better quality but there are still severe limitations. For transpositions in the order of a semitone, almost no objectionable alteration of the sounds can be heard. As the transposition ratio grows larger, in the practical range of plus or minus 2 octaves, the timbre of the output sound obtains a character that is specific to the harmonizer.

This modification can be heard both in the frequency domain and in the time domain and is due to the modulation of the signal by the chopping window. The spectrum of the input signal is indeed convolved with that of the window. The time-domain modulation can be characterized by its rate and by the spectrum of the window, which is dependent on its shape and its size. The longer the window, the lower the rate and hence the narrower the spectrum of the window and the less disturbing the modulation. The effect of a trapezoidal window will be stronger than that of a smoother one, such as the raised cosine window.

On the other hand, a larger window tends to deliver, through the overlap-add process, audible iterated copies of the input signals. For the transposition of percussive sounds, it is necessary to reduce the size of the window. Furthermore, to accurately replay transients and not smooth them out, the window should have sharp transitions. We see that a trade-off between audible spectral modulation and iterated transients has to be found for each type of sound. Musicians using the computer as a musical instrument might exploit these peculiarities in the algorithm to give their sound a unique flavor.

7.4.2 Pitch Shifting by Time Stretching and Resampling

The variable speed replay discussed in section 7.2 leads to a compression or expansion of the duration of a sound and to a pitch shift. This is accomplished by resampling in the time domain. Figure 7.1 illustrates the discrete-time signals and the corresponding spectra. The spectrum of the sound is compressed or expanded over the frequency axis. The harmonic relations

$$f_i = i \cdot f_{\text{fundamental}} \quad (7.4)$$

of the sound are not altered but are scaled according to

$$f_i^{\text{new}} = \alpha \cdot f_i^{\text{old}}. \quad (7.5)$$

The amplitudes of the harmonics remain the same $a_i^{\text{new}} = a_i^{\text{old}}$. In order to rescale the pitch shifted sound towards the original length a further time stretching algorithm can be applied to the sound. The result of pitch shifting followed by a time stretching algorithm is illustrated in Fig. 7.12.

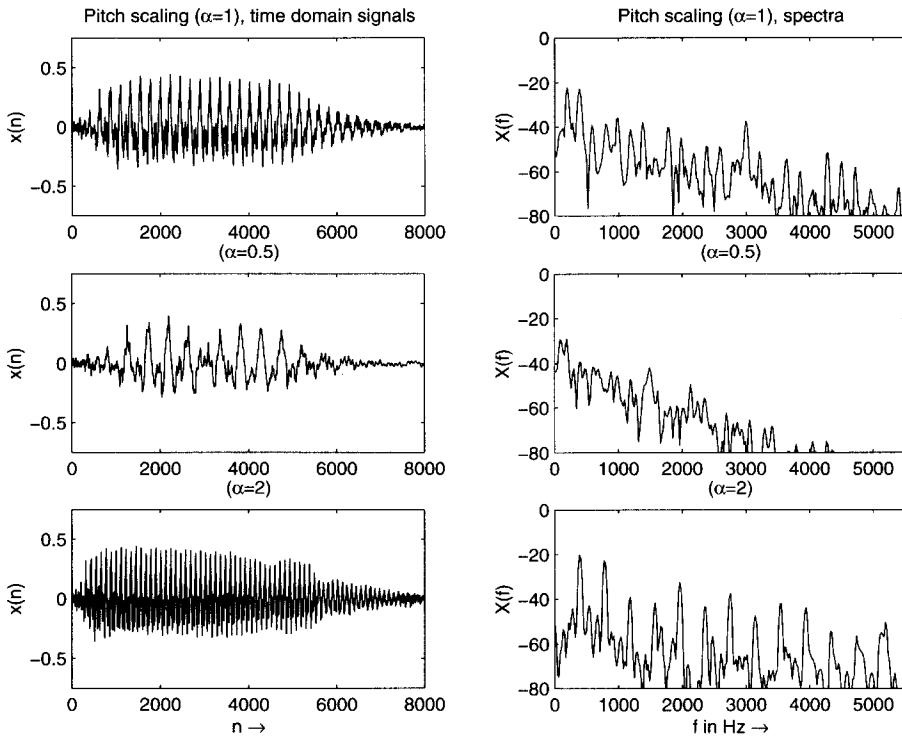


Figure 7.12 Pitch shifting followed by time correction.

The order of pitch shifting and time scaling can be changed, as shown in Fig. 7.13. First, a time scaling algorithm expands the input signal from length N_1 to length N_2 . Then a resampling operation with the inverse ratio N_1/N_2 performs pitch shifting and a reduction of length N_2 back to length N_1 .



Figure 7.13 Pitch shifting by time scaling and resampling.

The following M-file 7.3 demonstrates the implementation of the SOLA time scaling and pitch scaling algorithm:

M-file 7.3 (PitchScaleSOLA.m)
`% PitchScaleSOLA.m`

```

% Parameters:
%   analysis hop size      Sa = 256 (default parameter)
%   block length           N  = 2048 (default parameter)
%   pitch scaling factor   0.25 <= alpha <= 2
%   overlap interval        L  = 256*alpha/2
clear all,close all
[signal,Fs] = wavread('x1.wav');
DAFx_in = signal';

Sa=256;N=2048;          % time scaling parameters
M=ceil(length(DAFx_in)/Sa);

n1=512;n2=256;          % pitch scaling n1/n2
Ss=round(Sa*n1/n2);
L=256*(n1/n2)/2;

DAFx_in(M*Sa+N)=0;
Overlap=DAFx_in(1:N);

% ***** Time Stretching with alpha=n2/n1*****
..... % include main loop TimeScaleSOLA.m
% ***** End Time Stretching *****

% ***** Pitch shifting with alpha=n1/n2 *****
lfen=2048;lfen2=lfen/2;
w1=hanningz(lfen);w2=w1;

% for linear interpolation of a grain of length lx to length lfen
lx=floor(lfen*n1/n2);
x=1+(0:lfen-1)'*lx/lfen;
ix=floor(x);ix1=ix+1;
dx=x-ix;dx1=1-dx;
%
lmax=max(lfen,lx);
Overlap=Overlap';
DAFx_out=zeros(length(DAFx_in),1);

pin=0;pout=0;
pend=length(Overlap)-lmax;
% Pitch shifting by resampling a grain of length lx to length lfen
while pin<pend
    grain2=(Overlap(pin+ix).*dx1+Overlap(pin+ix1).*dx).* w1;
    DAFx_out(pout+1:pout+lfen)=DAFx_out(pout+1:pout+lfen)+grain2;
    pin=pin+n1;pout=pout+n2;
end;

```

7.4.3 Pitch Shifting by Delay Line Modulation

Pitch shifting or pitch transposing based on block processing is described in several publications. In [BB89] a pitch shifter based on an overlap add scheme with two time-varying delay lines is proposed (see Fig. 7.14). A cross-fade block combines the outputs of the two delay lines according to a cross-fade function. The signal is divided in small chunks. The chunks are read faster to produce higher pitches or slower to produce lower pitches. In order to produce a continuous signal output, two chunks are read simultaneously with a time delay equal to one half of the block length. A cross-fade is made from one chunk to the other at each end of a chunk [WG94, pp. 257-259].

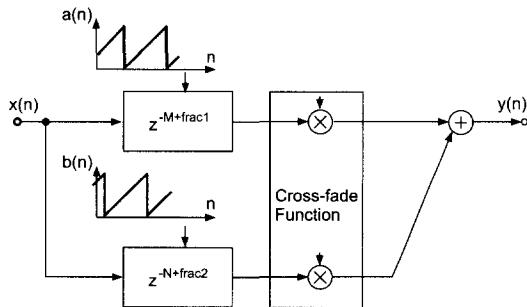


Figure 7.14 Pitch shifting.

The length of the delay lines is modulated by a sawtooth-type function. A similar approach is proposed in [Dat87] where the same configuration is used for time compression and expansion. A periodicity detection algorithm is used for calculating the cross-fade function in order to avoid cancellations during the cross-fades.

An enhanced method for transposing audio signals is presented in [DZ99]. The method is based on an overlap-add scheme and does not need any fundamental frequency estimation. The difference from other applications is the way the blocks are modulated and combined to the output signal. The enhanced transposing system is based on an overlap-add scheme with three parallel time-varying delay lines.

Figure 7.15 illustrates how the input signal is divided into blocks, which are resampled (phase modulation with a ramp type signal), amplitude modulated and summed yielding an output signal of the same length as the input signal. Adjacent blocks overlap with 2/3 of the block length.

The modulation signals form a system of three 120°-phase shifted raised cosine functions. The sum of these functions is constant for all arguments. Figure 7.16 also shows the topology of the pitch transposer. Since a complete cosine is used for modulation, the perceived sound quality of the processed signal is much better than in simple twofold overlap-add applications using several windows. The amplitude modulation only produces sum and difference frequencies with the base frequency of the modulation signal, which can be very low (6–10 Hz). Harmonics are not present

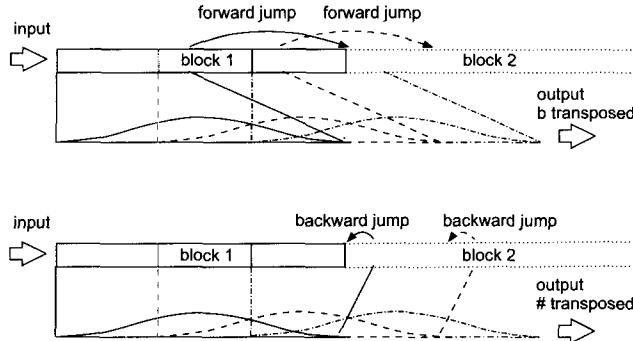


Figure 7.15 Pitch transposer: block processing, time shifting and overlap-add.

in the modulation signal and hence cannot form sum or difference frequencies of higher order. The perceived artifacts are phasing-like effects and are less annoying than local discontinuities of other applications based on twofold overlap-add methods.

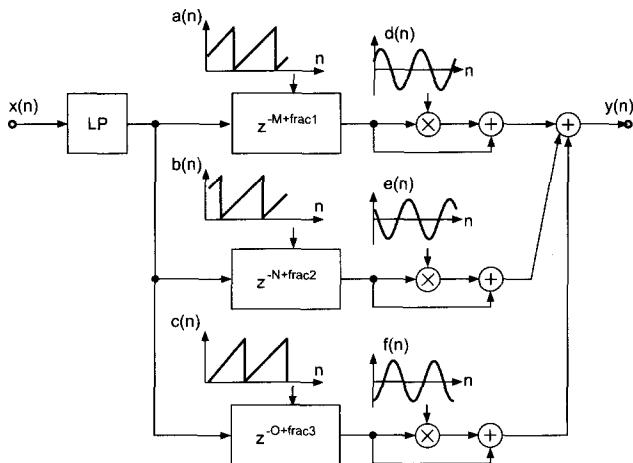


Figure 7.16 Pitch transposer: block diagram.

If we want to change the pitch of a signal controlled by another signal or signal envelope, we can also make use of delay line modulation. The effect can be achieved by performing a phase modulation of the recorded signal according to $y(n) = x(n - D(n))$. The modulating factor $D(n) = M + \text{DEPTH} \cdot x_{\text{mod}}(n)$ is now dependent on a modulating signal $x_{\text{mod}}(n)$. With this approach the pitch of the input signal $x(n)$ is changed according to the envelope of the modulating signal (see Fig. 7.17).

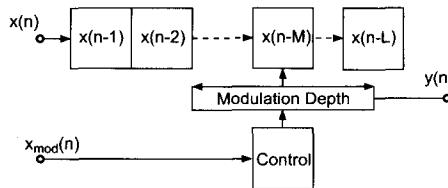


Figure 7.17 Pitch controlled by envelope of signal $x_{\text{mod}}(n)$.

7.4.4 Pitch Shifting by PSOLA and Formant Preservation

This technique is the dual operation to resampling in time domain, but in this case a resampling of the short-time spectral envelope is performed. The short-term spectral envelope describes a frequency curve going through all amplitudes of the harmonics. This is demonstrated in Fig. 7.18, where the spectral envelope is shown. The harmonics are again scaled according to $f_i^{\text{new}} = \beta \cdot f_i^{\text{old}}$, but the amplitudes of the harmonics $a_i^{\text{new}} = \text{env}(f_i^{\text{new}}) \neq a_i^{\text{old}}$ are now determined by sampling the spectral envelope. Some deviations of the amplitudes from the precise envelope can be noticed. This depends on the chosen pitch shifting algorithm.

The PSOLA algorithm can be conveniently used for pitch shifting a voice sound maintaining the formant position, and thus the vowel identity [ML95, BJ95]. The basic idea consists of time stretching the position of pitch marks, while the segment waveform is not changed. The underlining signal model of speech production is a pulse train filtered by a time varying filter corresponding to the vocal tract. The input segment corresponds to the filter impulse response and determines the formant position. Thus, it should not be modified. Conversely, the pitch mark distance determines the speech period, and thus should be modified accordingly. The aim of PSOLA analysis is to extract the local filter impulse response. As can be seen in Fig. 7.19, the spectrum of a segment extracted using a Hanning window with a length of two periods approximates the local spectral envelope. Longer windows tend to resolve the fine line structure of the spectrum, while shorter windows tend to blur the formant structure of the spectrum. Thus if we do not stretch the segment, the formant position is maintained. The operation of overlapping the segments at the new pitch mark position will resample the spectral envelope at the desired pitch frequency. When we desire a pitch shift by a factor β , defined as the ratio of the local synthesis pitch frequency to the original one $\beta = \tilde{f}_0(\tilde{t})/f_0(t)$, the new pitch period will be given by $\tilde{P}(\tilde{t}) = P(t)/\beta$, where in this case $\tilde{t} = t$ because time is not stretched.

The analysis algorithm is the same as that previously seen for PSOLA time stretching in section 7.3.3 (see Fig. 7.9). The synthesis algorithm is modified (see Fig. 7.20) according to the following steps:

- for every synthesis pitch mark \tilde{t}_k
1. Choice of the corresponding analysis segment i (identified by the time mark t_i) minimizing the time distance $|t_i - \tilde{t}_k|$.

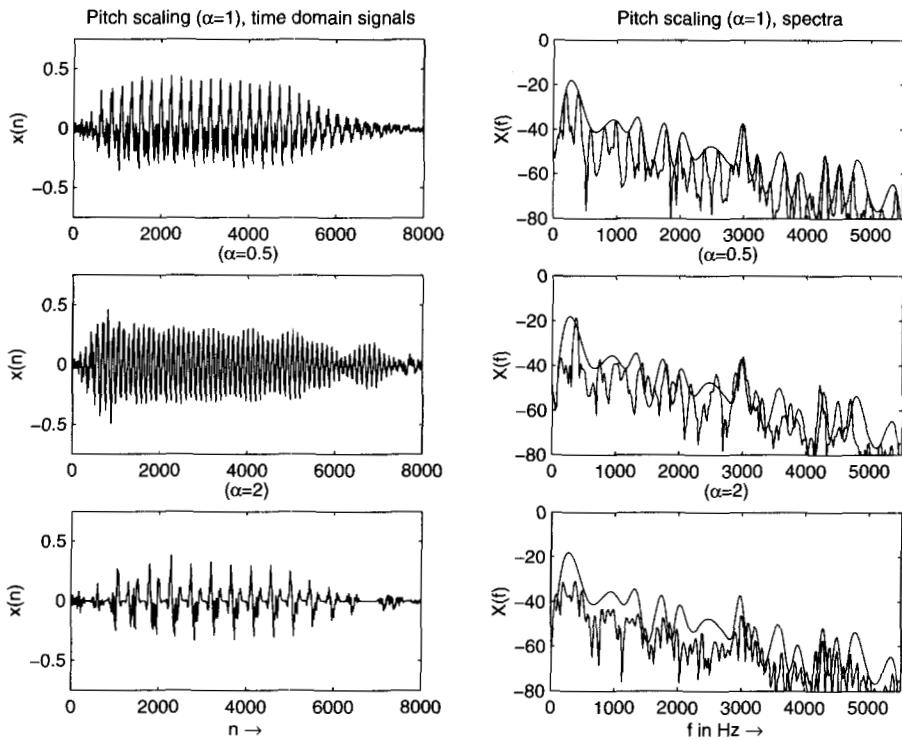


Figure 7.18 Pitch shifting by the PSOLA method: frequency resampling the spectral envelope.

2. Overlap and add the selected segment. Notice that some input segments will be repeated for $\beta > 1$ (higher pitch) or discarded when $\beta < 1$ (lower pitch).
3. Determination of the time instant \tilde{t}_{k+1} where the next synthesis segment will be centered, in order to preserve the local pitch, by the relation

$$\tilde{t}_{k+1} = \tilde{t}_k + \tilde{P}(\tilde{t}_k) = \tilde{t}_k + P(t_i)/\beta.$$

- for large pitch shifts, it is advisable to compensate the amplitude variation, introduced by the greater or lesser overlapping of segments, by multiplying the output signal by $1/\beta$.

It is possible to combine time stretching by a factor α with pitch shifting. In this case for every synthesis pitch mark \tilde{t}_k the first step of the synthesis algorithm above presented will be modified as choice of the corresponding analysis segment i (identified by the time mark t_i) minimizing the time distance $|\alpha t_i - \tilde{t}_k|$.

The PSOLA algorithm is very effective for speech processing and is computationally very efficient, once the sound has been analyzed, so it is widely used for speech

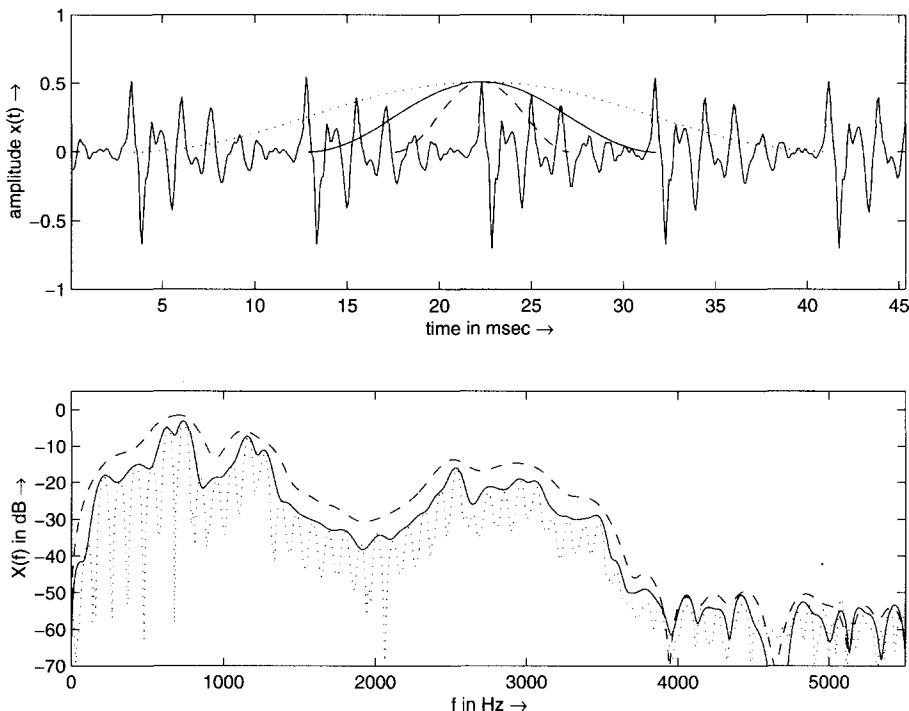


Figure 7.19 Spectrum of segments extracted from a vowel /a/ by using a Hanning window respectively long 4 (dotted line), 2 (solid line), and 1 (dashed line) pitch periods. It can be noticed that the solid line approximates the local spectral envelope.

synthesis from a database of diphones, for prosody modification, for automatic answering machines etc. For wide variation of the pitch it presents some artifacts. On the other hand the necessity of a preliminary analysis stage for obtaining a pitch contour makes the real-time implementation of an input signal modification difficult. Also the estimation of glottal pulses can be difficult. A solution is to place the pitch marks at a pitch synchronous rate, regardless of the true position of the glottal pulses. The resulting synthesis quality will be only slightly decreased (see for example Fig. 7.21).

A further effect that can be obtained by a variation of PSOLA is linear scaling of formant frequencies (see Fig. 7.22). In fact, we saw that a time scale of a signal corresponds to an inverse frequency scale. Thus when we perform time scaling of the impulse response of a filter, we inversely scale the frequency of formants. In PSOLA terms, this corresponds to time scaling the selected input segments before overlap and add in the synthesis step, without any change in the pitch marks calculation. To increase the frequencies of formants by a factor γ , every segment should be shortened by a factor $1/\gamma$ by resampling. For example, the average formant frequencies of female adults are about 16 percent higher than those of male adults, and children's

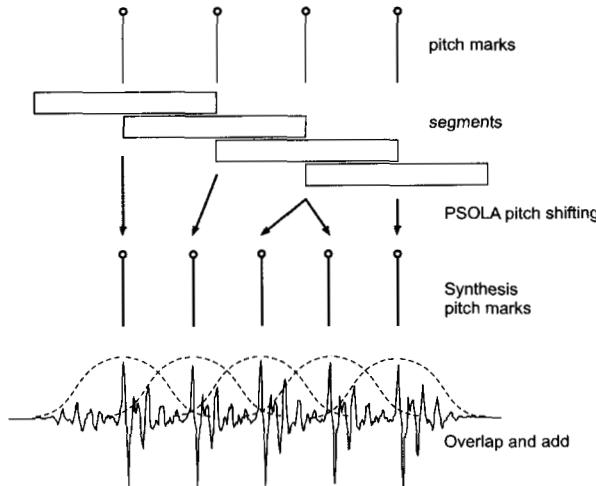


Figure 7.20 PSOLA: synthesis algorithm for pitch shifting.

formants are about 20 percent higher than female formants. Notice that care should be taken when the frequencies increase in order to avoid foldover. Ideally band-limited resampling should be used.

The following M-file 7.4 shows the implementation of the basic PSOLA synthesis algorithm. It is based on the PSOLA time stretching algorithm shown in section 7.3.3.

```
M-file 7.4 (psolaf.m)
function out=psolaf(in,m,alpha,beta,gamma)
%
%      gamma newFormantFreq/oldFormantFreq
%
%      the internal loop as
tk = P(1)+1; %output pitch mark
while round(tk)<Lout
    [minimum i]=min(abs(alpha*m-tk)); % find analysis segment
    pit=P(i);pitStr=floor(pit/gamma);
    gr=in(m(i)-pit:m(i)+pit).*hanning(2*pit+1);
    gr=interp1(-pit:1:pit,gr,-pitStr*gamma:gamma:pit);% stretch segm.
    iniGr=round(tk)-pitStr;endGr=round(tk)+pitStr;
    if endGr>Lout, break; end
    out(iniGr:endGr)=out(iniGr:endGr)+gr; % overlap new segment
    tk=tk+pit/beta;
end % end of while
```

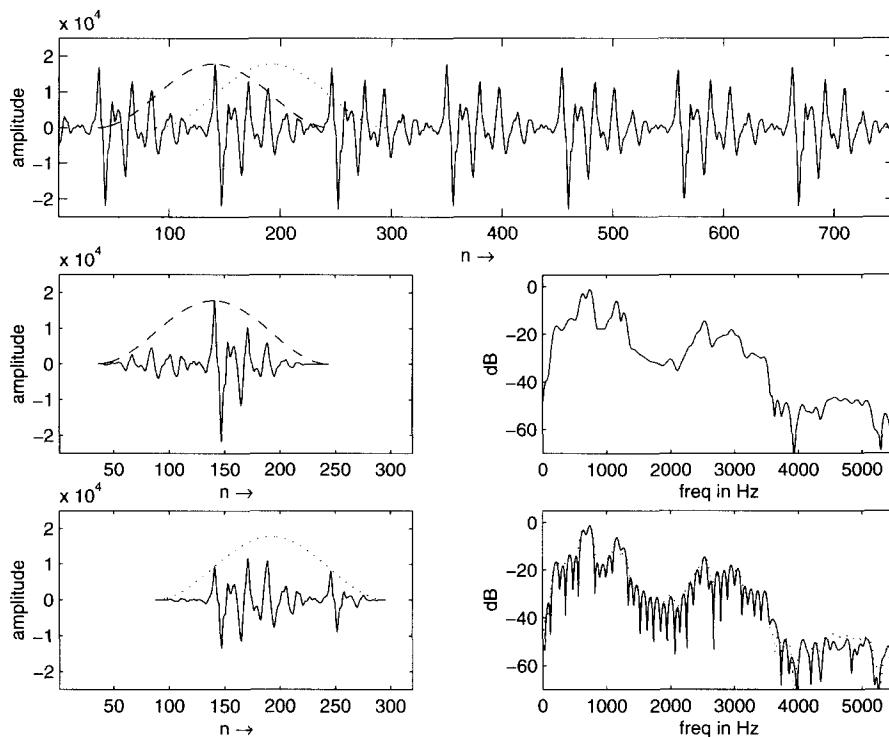


Figure 7.21 Comparison of a segment extracted in the correspondence with glottal pulse with one extracted between pitch pulses.

7.5 Time Shuffling and Granulation

7.5.1 Time Shuffling

Introduction

Musique concrète has made intensive use of splicing of tiny elements of magnetic tape. When mastered well, this assembly of hundreds of fragments of several tens of milliseconds allows an amalgamation of heterogeneous sound materials, at the limit of the time discrimination threshold. This manual operation called micro-splicing was very time-consuming. Bernard Parmegiani suggested in 1980 at the Groupe de Recherches Musicales (GRM) that this could be done by computers. An initial version of the software was produced in the early eighties. After being rewritten, improved and ported several times, it was eventually made available on personal computers in the form of the program called *brassage* in French that will be translated here as *time shuffling* [Ges98, Ges00].

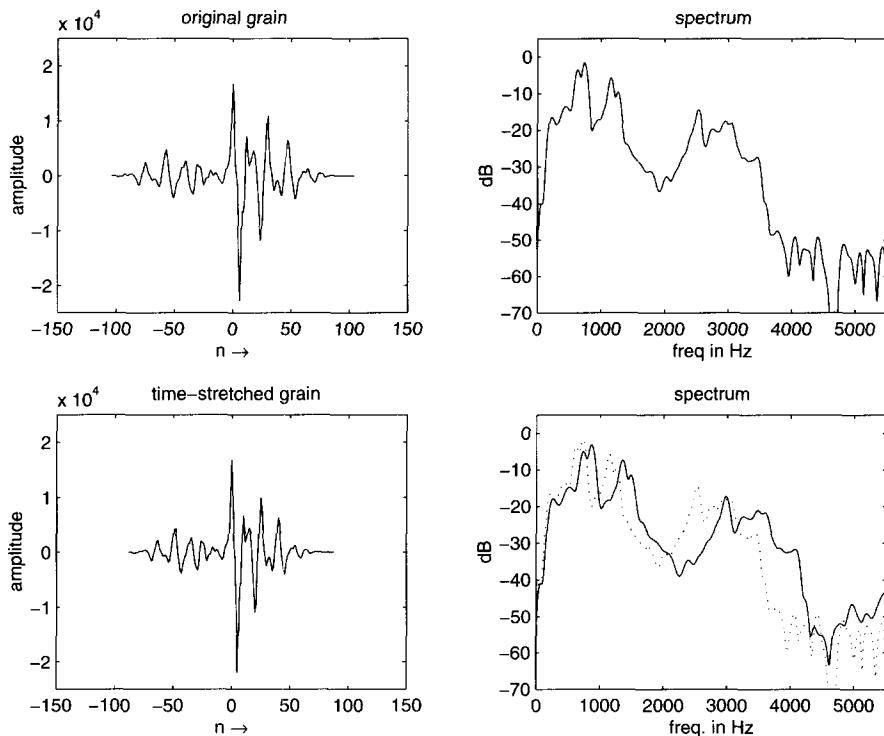


Figure 7.22 PSOLA: variation of PSOLA as linear formant scaling.

Signal Processing

Let us describe here an elementary algorithm for time shuffling that is based on the superposition of two time segments that are picked randomly from the input signal (see Fig. 7.23):

1. Let $x(n)$ and $y(n)$ be the input and output signals.
2. Specify the duration d of the fragments and the duration $D \geq d$ of the time period $[n - D, n]$ from which the time segments will be selected.
3. Store the incoming signal $x(n)$ in a delay line of length D .
4. Choose at random the delay time τ_1 with $d \leq \tau_1 \leq D$.
5. Select the signal segment x_{1d} of duration d beginning at $x(n - \tau_1)$.
6. Follow the same procedure (steps 4 and 5) for a second time segment x_{2d} .
7. Read x_{1d} and x_{2d} and apply an amplitude envelope W to each of them in order to smooth out the discontinuities at the borders.

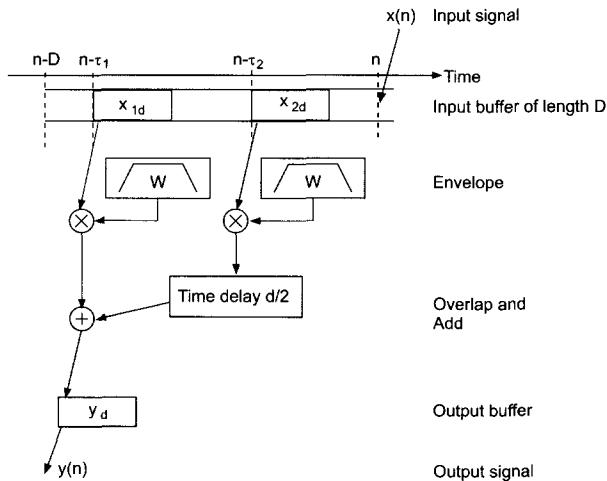


Figure 7.23 Time shuffling: 2 input segments, selected at random from the past input signal, are overlap-added to produce an output time segment. When one of the input segment is finished, a new one is selected.

8. When the reading of x_{1d} or x_{2d} is finished, iterate the procedure for each of them.
9. Compute the output as the overlap add of the sequence of x_{1d} and x_{2d} with a time shift of $d/2$.

Musical Applications and Control

The version described above introduces local disturbances into the signal's actual timing, while preserving the overall continuity of its time sequence. Many further refinements of this algorithm are possible. A random amplitude coefficient could be applied to each of the input segments in order to modify the density of the sound material. The shape of the envelope could be modified in order to retain more of the input time structure or, on the contrary, to smooth it out and blend different events with each other. The replay speed of the segments could be varied in order to produce transposition or glissandi.

At a time when computer tools were not yet available, Bernard Parmegiani magnificently illustrated the technique of tape-based micro-splicing in works such as "Violostries" (1964) or "Dedans-Dehors" (1977) [m-Par64, m-Par77]. The elementary algorithm presented above can be operated in real time but other off-line versions have also been implemented which offered many more features. They have the ability to merge fragments of any size, sampled from a random field, also of any dimension: from a few samples to several minutes. Thus apart from generating fusion phenomena, for which the algorithm was conceived, the software was able to produce cross-fading of textured sound and other sustained chords, infinitely small

variations in signal stability, interpolation of fragments with silence or sounds of other types [Ges98]. Jean-Claude Risset used this effect to perform sonic developments from short sounds, such as stones and metal chimes [m-INA3, Sud-I, 3'44" to 4'38"];[Ris98, Ges00] and to produce a “stuttering” piano, further processed by ring modulation [m-INA3, Sud-I, 4'30", 5'45"]. Starting from “found objects” such as bird songs, he rearranged them in a compositional manner to obtain first a pointilistic rendering, then a stretto-like episode [m-INA3, Sud-I, 1'42" to 2'49"].

7.5.2 Granulation

Introduction

In the previous sections about pitch shifting and time stretching we have proposed algorithms that have limitations as far as their initial purpose is concerned. Beyond a limited range of modification of pitch or of time duration, severe artifacts appear. The time shuffling method considers these artifacts from an artistic point of view and takes them for granted. Out of the possibilities offered by the methods and by their limitations, it aims to create new sound structures. Whereas the time shuffling effect exploits the possibilities of a given software arrangement, which could be considered here as a “musical instrument”, the idea of building a complex sound out of a large set of elementary sounds could find a larger framework.

The physicist Dennis Gabor proposed in 1947 the idea of the quantum of sound, an indivisible unit of information from the psychoacoustical point of view. According to his theory, a granular representation could describe any sound. Granular synthesis was first suggested as a computer music technique for producing complex sounds by Iannis Xenakis (1971) and Curtis Roads (1978). This technique builds up acoustic events from thousands of sound grains. A sound grain lasts a brief moment (typically 1 to 100 ms), which approaches the minimum perceivable event time for duration, frequency, and amplitude discrimination [Roa96, Roa98, Tru00a].

The granulation effect is an application of granular synthesis where the material out of which the grains are formed is an input signal. Barry Truax has developed this technique [Tru88, Tru94] by a first real-time implementation and using it extensively in his compositional pieces.

Signal Processing

Let $x(n)$ and $y(n)$ be the input and output signals. The grains $g_k(i)$ are extracted from the input signal with the help of a window function $w_k(i)$ of length L_k by

$$g_k(i) = x(i + i_k)w_k(i) \quad (7.6)$$

with $i = 0, \dots, L_{k-1}$. The time instant i_k indicates the point where the segment is extracted; the length L_k determines the amount of signal extracted; the window waveform $w_k(i)$ should ensure fade-in and fade-out at the border of the grain and affects the frequency content of the grain. Long grains tend to maintain the timbre

identity of the portion of the input signal, while short ones acquire a pulse-like quality. When the grain is long, the window has a flat top and it used only to fade-in and fade-out the borders of the segment.

The following M-files 7.5 and 7.6 show the extraction of short and long grains:

M-file 7.5 (grainSh.m)

```
function y = grainSh(x,init,L)
% extract a short grain
% x      input signal
% init   first sample
% L      grain length (in samples)
y=x(init:init+L-1).*hanning(L');
```

M-file 7.6 (grainLn.m)

```
function y = grainLn(x,iniz,L,Lw)
% extract a long grain
% x      input signal
% init   first sample
% L      grain length (in samples)
% Lw     length fade-in and fade-out (in samples)
if length(x) <= iniz+L , error('length(x) too short.'), end
y = x(iniz:iniz+L-1); % extract segment
w = hanning(2*Lw+1)';
y(1:Lw) = y(1:Lw).*w(1:Lw); % fade-in
y(L-Lw+1:L) = y(L-Lw+1:L).*w(Lw+2:2*Lw+1); % fade-out
```

The synthesis formula is given by

$$y(n) = \sum_k a_k g_k(n - n_k) \quad (7.7)$$

where a_k is an eventual amplitude coefficient and n_k is the time instant where the grain is placed in the output signal. Notice that the grains can overlap. To overlap a grain g_k (grain) at instant $n_k = (\text{iniOLA})$ with amplitude a_k , the following MATLAB instructions can be used

```
endOLA = iniOLA+length(grain)-1;
y(iniOLA:endOLA) = y(iniOLA:endOLA) + ak * grain;
```

An example of granulation with random values of the parameters grain initial point and length, output point and amplitude is shown in Fig. 7.24. The M-file 7.7 shows the implementation of the granulation algorithm.

M-file 7.7 (granulation.m)

```
% granulation.m
f=fopen('a_male.m11');
x=fread(f,'int16');
```

```

fclose(f);

Ly=length(x); y=zeros(1,Ly); %output signal
% Constants
nEv=4; maxL=200; minL=50; Lw=20;
% Initializations
L = round((maxL-minL)*rand(1,nEv))+minL; %grain length
initIn = ceil((Ly-maxL)*rand(1,nEv)); %init grain
initOut= ceil((Ly-maxL)*rand(1,nEv)); %init out grain
a = rand(1,nEv); %ampl. grain
endOut=initOut+L-1;
% Synthesis
for k=1:nEv,
    grain=grainLn(x,initIn(k),L(k),Lw);
    y(initOut(k):endOut(k))=y(initOut(k):endOut(k))+a(k)*grain;
end

```

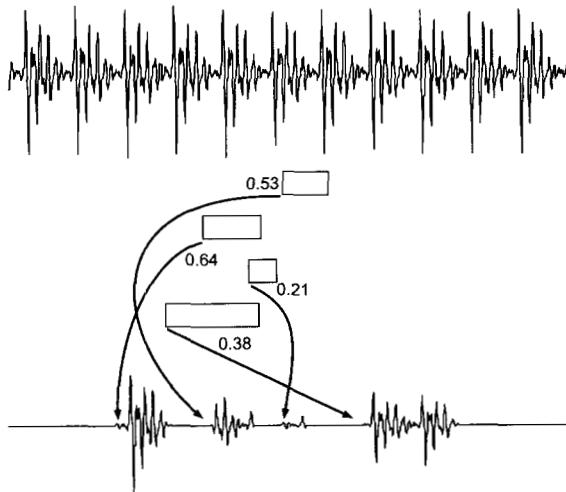


Figure 7.24 Example of granulation.

This technique is quite general and can be employed to obtain very different sound effects. The result is greatly influenced by the criterion used to choose the instants n_k . If these points are regularly spaced in time and the grain waveform does not change too much, the technique can be interpreted as a filtered pulse train, i.e. it produces a periodic sound whose spectral envelope is determined by the grain waveform interpreted as impulse response. An example is the PSOLA algorithm shown in the previous sections 7.3.3 and 7.4.4. When the distance between two subsequent grains is much greater than L_k , the sound will result in grains separated by interruptions or silences, with a specific character. When many short grains overlap (i.e. the distance is less than L_k), a sound texture effect is obtained.

The strategies for choosing the synthesis instants can be grouped into two rather simplified categories: synchronous, mostly based on deterministic functions; and asynchronous, based on stochastic functions. Grains can be organized in streams. There are two main control variables: the delay between grains for a single stream, and the degree of synchronicity among grains in different streams. Given that the local spectrum affects the global sound structure, it is possible to use input sounds that can be parsed in grains without altering the complex characteristics of the original sound, as water drops for stream-like sounds.

It is further possible to modify the grain waveform with a time transformation, such as modulation for frequency shifting or time stretching for frequency scaling [DP91]. The main parameters of granulation are: grain duration, selection order from input sound, amplitude of grains, temporal pattern in synthesis, grain density (i.e. grains per second). Density is a primary parameter, as it determines the overall texture, whether sparse or continuous. Notice that it is possible to extract grains from different sound files to create hybrid textures, e.g. evolving from one texture to another.

Musical Applications

A demonstration of the effect is provided in [m-Tod99]. Further examples can be found in [m-Wis94c]. Barry Truax has used the technique of granulation to process sampled sound as compositional material. In “The Wings of Nike” (1987) he has processed only short “phonemic” fragments but longer sequences of environmental sound have been used in pieces such as “Pacific” (1990). In each of these works, the granulated material is time stretched by various amounts and thereby produces a number of perceptual changes that seem to originate from within the sound [Tru00b, m-Tru95].

In “Le Tombeau de Maurice”, Ludger Brümmer uses the granulation technique in order to perform timbral, rhythmic as well as harmonic modifications [m-Bru97]. A transition from the original sound color of an orchestral sample towards noise pulses is achieved by reducing progressively the size of the grains. At an intermediate grain size, the pitch of the original sound is still recognizable although the time structure has already disappeared [m-Bru97, 3'39"-4'12"]. A melody can be played by selecting grains of different pitches and by varying the tempo at which the grains are replayed [m-Bru97, 8'38"-9'10"]. New melodies can even appear out of a two-stage granulation scheme. A first series of grains is defined out of the original sample whereas the second is a granulation of the first one. Because of the stream segregation performed by the hearing system, the rhythmic as well as the harmonic grouping of the grains is constantly evolving [m-Bru97, 9'30"-10'33"].

7.6 Conclusion

The effects described in this chapter are based on the division of the input sound into short segments. These segments are processed by simple methods such as time

scaling by resampling or amplitude multiplication by an envelope. The segment waveform is not changed, thus maintaining the characteristic of the source signal.

Two categories of effects can be obtained, depending on the strategy used to place the segments in time during the synthesis. If the order and organization of extracted segments are carefully maintained, time stretching or pitch shifting can be performed. Basic methods, SOLA and PSOLA, are presented and their characteristics are discussed. These effects aim to produce sounds that are perceived as similar to the original, but are modified in duration or pitch. As often happens with digital audio effects, the artifacts produced by these methods can be used as a method for deformation of the input sound, whilst maintaining its main characteristics. The low computational complexity of time-segment processing allows efficient real-time applications. Nevertheless, these algorithms produce artifacts that limit their scope of application. More advanced methods for time stretching and pitch shifting will be introduced in Chapters 8-11.

The second category changes the organization and the order of the segments to a great extent, and thus leads to time shuffling and granulation. In this case, the input sound can be much less recognizable. The central element becomes the grain with its amplitude envelope and time organization. These techniques can produce results from sparse grains to dense textures, with a very loose relationship with the original sound. It should be noticed that the wide choice of strategies for grain organization implies a sound composition attitude from the user. Thus granulation became a sort of metaphor for music composition starting from the microlevel.

Sound and Music

- [m-Bru93] L. Brümmer: The Gates of H. Computer music. CCRMA 1993. In: CRI, "The listening room", CD edel 0014522TLR, Hamburg, 1995.
- [m-Bru97] L. Brümmer: Le Tombeau de Maurice, for computer-generated tape. In: Computer music @ CCRMA. CD CCRMAV02, 1997.
- [m-Eim62] H. Eimert: Epitaph für Aikichi Kuboyama. Electronic music composition, 1962. Studio-Reihe neuer Musik, Wergo, LP WER 60014. Reeditio-n as a CD, Koch/Schwann, 1996.
- [m-Fur93] K. Furukawa: Swim, Swan, composition for clarinet and live-electronics. ZKM, 1993.
- [m-Fur97] K. Furukawa: Den ungeborenen Göttern. Multimedia-Opera, ZKM, 1997.
- [m-INA3] J.-C. Risset: Sud, Dialogues, Inharmonique, Mutations. CD INA C1003.
- [m-Par64] B. Parmegiani: Violostries, 1964. IDEAMA CD 051 Target Collection, ZKM and CCRMA, 1996.

- [m-Par77] B. Parmegiani: *Dedans-Dehors*, 1977. INA-GRM.
- [m-Sch98] P. Schaeffer and G. Reibel: *Solfège de l'objet sonore*. Booklet + 3 CDs. First published 1967. INA-GRM, 1998.
- [m-Tod99] T. Todoroff: Real-time granulation. *Revenant.aif*, *Reven2G.aif*, *RevenCmb.aif*. DAFX-Sound Library.
- [m-Tru95] B. Truax: Granular Time-shifting and Transposition Composition Examples. In *Computer Music Journal*, Volume 19 Compact Disc, Index 6, 1995.
- [m-Wis94c] T. Wishart: Audible design, Sound examples. CD. *Orpheus the Pantomime*, York, 1994.

Bibliography

- [And95] C. Anderton. *Multieffects for Musicians*. Amsco Publications, 1995.
- [BB89] K. Bogdanowicz and R. Blecher. Using multiple processors for real-time audio effects. In *AES 7th International Conference*, pp. 337–342, 1989.
- [BJ95] R. Bristow-Johnson. A detailed analysis of a time-domain formant-corrected pitch shifting algorithm. *J. Audio Eng. Soc.*, 43(5):340–352, 1995.
- [Car92] T. Cary. *Illustrated Compendium of Musical Technology*. Faber and Faber, 1992.
- [Chi82] M. Chion. *La musique électroacoustique*. QSJ No 1990, PUF, 1982.
- [CR83] R.E. Crochiere and L.R. Rabiner. *Multirate Digital Signal Processing*. Prentice-Hall, 1983.
- [Dat87] J. Dattoro. Using digital signal processor chips in a stereo audio time compressor/expander. In *Proc. 83rd AES Convention*, Preprint 2500, 1987.
- [DP91] G. De Poli and A. Piccialli. Pitch-synchronous granular synthesis. In G. De Poli, A. Piccialli, and C. Roads (eds), *Representations of Musical Signals*, pp. 187–219. MIT Press, 1991.
- [Dut88] P. Dutilleux. *Mise en œuvre de transformations sonores sur un système temps-réel*. Technical report, Rapport de stage de DEA, CNRS-LMA, June 1988.
- [DZ99] S. Disch and U. Zölzer. Modulation and delay line based digital audio effects. In *Proc. DAFX-99 Digital Audio Effects Workshop*, pp. 5–8, Trondheim, December 1999.

- [End97] B. Enders. *Lexikon Musikelektronik*. Atlantis Schott, 1997.
- [Gas87] P.S. Gaskell. A hybrid approach to the variable speed replay of digital audio. *J. Audio Eng. Soc.*, 35:230–238, April 1987.
- [Ges98] Y. Geslin. Sound and music transformation environments: A twenty-year experiment at the “Groupe de Recherches Musicales”. In *Proc. DAFX-98 Digital Audio Effects Workshop*, pp. 241–248, Barcelona, November 1998.
- [Ges00] Y. Geslin. About the various types of Phonogènes. GRM, Personal communication, 2000.
- [GRH73] T.A. Giordano, H.B. Rothman, and H. Hollien. Helium speech unscramblers — a critical review of the state of the art. *IEEE Trans Audio and Electroacoustics*, AU-21(5), October 1973.
- [Hal95] H.P. Haller. *Das Experimental Studio der Heinrich-Strobel-Stiftung des Südwestfunks Freiburg 1971–1989, Die Erforschung der Elektronischen Klangumformung und ihre Geschichte*. Nomos, 1995.
- [HMC89] C. Hamon, E. Moulines, and F. Charpentier. A diphone synthesis system based on time-domain prosodic modifications of speech. In *Proc. ICASSP*, pp. 238–241, 1989.
- [Lar98] J. Laroche. Time and pitch scale modifications of audio signals. In M. Kahrs and K.-H. Brandenburg (eds), *Applications of Digital Signal Processing to Audio and Acoustics*, pp. 279–309. Kluwer, 1998.
- [Lee72] F.F. Lee. Time compression and expansion of speech by the sampling method. *J. Audio Eng. Soc.*, 20(9):738–742, November 1972.
- [Mas98] D.C. Massie. Wavetable sampling synthesis. In M. Kahrs and K.-H. Brandenburg (eds), *Applications of Digital Signal Processing to Audio and Acoustics*, pp. 311–341. Kluwer, 1998.
- [MC90] E. Moulines and F. Charpentier. Pitch synchronous waveform processing techniques for text-to speech synthesis using diphones. *Speech Communication*, 9(5/6):453–467, 1990.
- [McN84] G.W. McNally. Variable speed replay of digital audio with constant output sampling rate. In *Proc. 76th AES Convention*, Preprint 2137, 1984.
- [MEJ86] J. Makhoul and A. El-Jaroudi. Time-scale modification in medium to low rate speech coding. In *Proc. ICASSP*, pp. 1705–1708, 1986.
- [ML95] E. Moulines and J. Laroche. Non-parametric technique for pitch-scale and time-scale modification of speech. *Speech Communication*, 16:175–205, 1995.
- [Mol60] A. Moles. *Les musiques expérimentales*. Trad. D. Charles. Cercle d’Art Contemporain, 1960.

- [Pou54] J. Poullin. L'apport des techniques d'enregistrement dans la fabraction de matières et formes musicales nouvelles. Applications à la musique concrète. *L'Onde Électrique*, 34(324):282–291, 1954.
- [PS57] J. Poullin and D.A. Sinclair. *The Application of Recording Techniques to the Production of New Musical Materials and Forms. Application to "Musique Concrète"*. National Research Council of Canada, 1957. Technical Translation TT-646, pp. 1–29.
- [Ris98] J.-C. Risset. Example of the musical use of digital audio effects. In *Proc. DAFX-99 Digital Audio Effects Workshop*, pp. 254–259, Trondheim, December 1998.
- [Roa96] C. Roads. *The Computer Music Tutorial*. MIT Press, 1996.
- [Roa98] C. Roads. Micro-sound, history and illusion. In *Proc. DAFX-98 Digital Audio Effects Workshop*, pp. 260–269, Barcelona, November 1998.
- [RW85] S. Roucos and A.M. Wilgus. High quality time-scale modification for speech. In *Proc. ICASSP*, pp. 493–496, 1985.
- [Sch73] P. Schaeffer. *La musique concrète*. QSJ No 1287, PUF 1973.
- [Spr55] A.M. Springer. Ein akustischer Zeitregler. *Gravesaner Blätter*, (1):32–27, July 1955.
- [Spr59] J.M. Springer. Akustischer Tempo- und Tonlagenregler. *Gravesaner Blätter*, (13):80, 1959.
- [Tru88] B. Truax. Real-time granular synthesis with a digital signal processor. *Computer Music Journal*, 12(2):14–26, 1988.
- [Tru94] B. Truax. Discovering inner complexity: time-shifting and transposition with a real-time granulation technique. *Computer Music Journal*, 18(2):28–38, 1994.
- [Tru00a] B. Truax. <http://www.sfu.ca/~truax/gran.html>. *Granular synthesis*, 2000.
- [Tru00b] B. Truax. <http://www.sfu.ca/~truax/gsample.html>. *Granulation of sampled sounds*, 2000.
- [WG94] M. Warstat and T. Görne. *Studiotechnik — Hintergrund und Praxiswissen*. Elektor-Verlag, 1994.
- [Whi99] P. White. *Creative Recording, Effects and Processors*. Sanctuary Publishing, 1999.

Chapter 8

Time-frequency Processing

D. Arfib, F. Keiler, U. Zölzer

8.1 Introduction

This chapter describes the use of time-frequency representations of signals in order to produce transformations of sounds. A very interesting (and intuitive) way of modifying a sound is to make a two-dimensional representation of it, modify this representation in some or other way and reconstruct a new signal from this representation (see Fig. 8.1). Consequently a digital audio effect based on time-frequency representations requires three steps: an analysis (sound to representation), a transformation (of the representation) and a resynthesis (getting back to a sound).

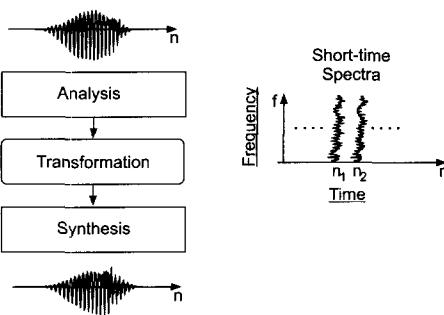


Figure 8.1 Digital audio effects based on analysis, transformation and synthesis (resynthesis).

The direct scheme of spectral analysis, transformation and resynthesis will be discussed in section 8.2. We will explore the modification of the magnitude $|X(k)|$ and phase $\varphi(k)$ of these representations before resynthesis. The analysis/synthesis

scheme is termed the *phase vocoder* (see Fig. 8.2). The input signal $x(n)$ is multiplied by a sliding window of finite length N , which yields successive windowed signal segments. These are transformed to the spectral domain by FFTs. In this way a time-varying spectrum $X(n, k) = |X(n, k)|e^{j\varphi(n, k)}$ with $k = 0, 1, \dots, N - 1$ is computed for each windowed segment. The short-time spectra can be modified or transformed for a digital audio effect. Then each modified spectrum is applied to an IFFT and windowed in the time domain. The windowed output segments are then overlapped and added yielding the output signal. It is also possible to complete this time-frequency processing by spectral processing, which is dealt with in the next two chapters.

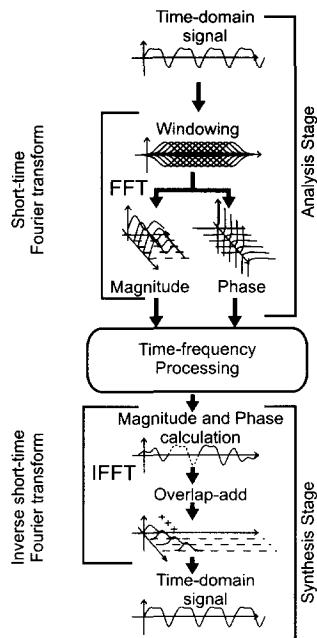


Figure 8.2 Time-frequency processing based on the phase vocoder: analysis, transformation and synthesis.

8.2 Phase Vocoder Basics

The concepts of short-time Fourier analysis and synthesis have been widely described in the literature [Por76, Cro80, CR83]. We will briefly summarize the basics and define our notation of terms for the application to digital audio effects.

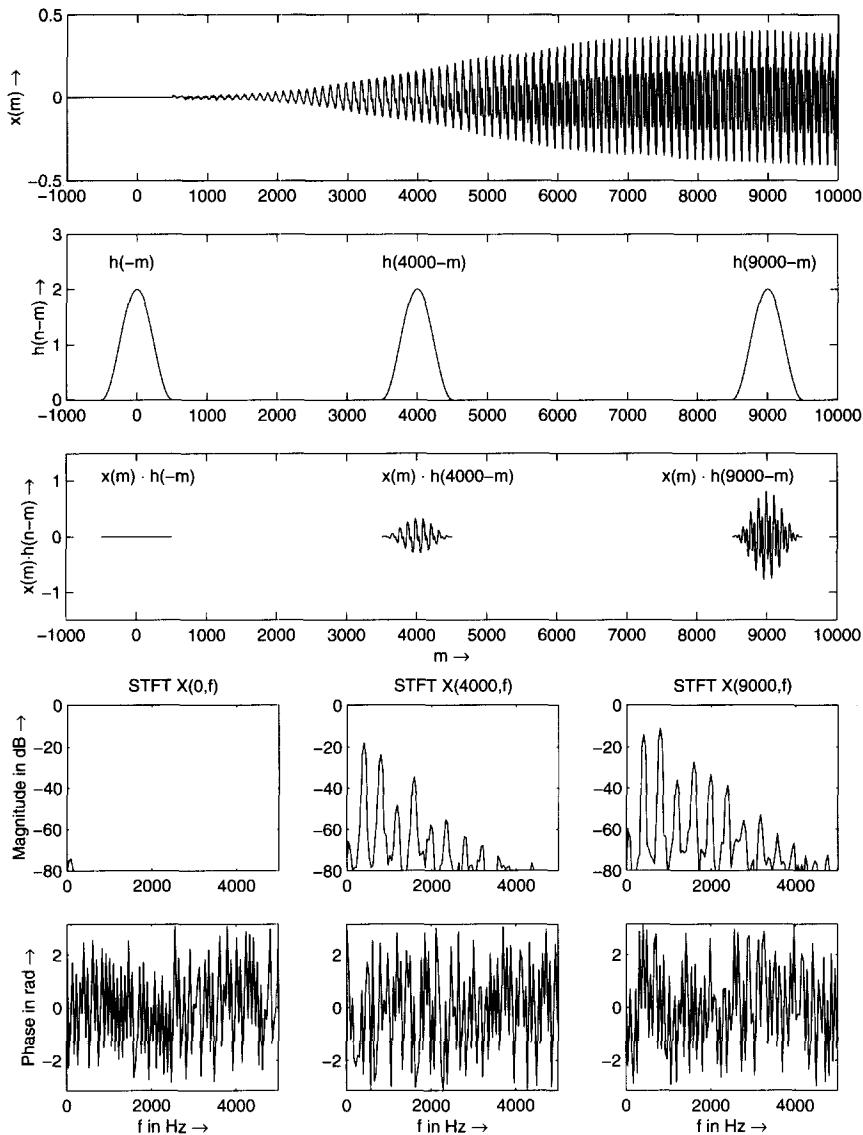


Figure 8.3 Sliding analysis window and short-time Fourier transform.

The short-time Fourier transform (STFT) of the signal $x(n)$ is given by

$$X(n, k) = \sum_{m=-\infty}^{\infty} x(m)h(n-m)W_N^{mk}, \quad k = 0, 1, \dots, N-1 \quad (8.1)$$

$$W_N = e^{-j2\pi/N} \quad (8.2)$$

$$= X_R(n, k) + jX_I(n, k) = |X(n, k)| \cdot e^{j\varphi(n, k)}. \quad (8.3)$$

$X(n, k)$ is a complex number and represents the magnitude $|X(n, k)|$ and phase $\varphi(n, k)$ of a time-varying spectrum with the frequency bin (index) $0 \leq k \leq N - 1$ and time index n . Note that the summation index is m in (8.1). At each time index n the signal $x(m)$ is weighted by a finite length window $h(n - m)$. Thus the computation of (8.1) can be performed by a finite sum over m with an FFT of length N . Figure 8.3 shows the input signal $x(m)$ and the sliding window $h(n - m)$ for three time indices of n . The middle plot shows the finite length windowed segments $x(m) \cdot h(n - m)$. These segments are transformed by the FFT yielding the short-time spectra $X(n, k)$ given by (8.1). The lower two rows in Fig. 8.3 show the magnitude and phase spectra of the corresponding time segments.

8.2.1 Filter Bank Summation Model

The computation of the time-varying spectrum of an input signal can also be interpreted as a parallel bank of N bandpass filters, as shown in Fig. 8.4, with impulse responses and Fourier transforms given by

$$h_k(n) = h(n)W_N^{-nk}, \quad k = 0, 1, \dots, N - 1 \quad (8.4)$$

$$H_k(e^{j\Omega}) = H(e^{j(\Omega - \Omega_k)}), \quad \Omega_k = \frac{2\pi}{N}k. \quad (8.5)$$

Each bandpass signal $y_k(n)$ is obtained by filtering the input signal $x(n)$ with the corresponding bandpass filter $h_k(n)$. Since the bandpass filters are complex-valued, we get complex-valued output signals $y_k(n)$, which will be denoted by

$$y_k(n) = \tilde{X}(n, k) = |X(n, k)| \cdot e^{j\tilde{\varphi}(n, k)}. \quad (8.6)$$

These filtering operations are performed by the convolutions

$$y_k(n) = \sum_{m=-\infty}^{\infty} x(m)h_k(n - m) = \sum_{m=-\infty}^{\infty} x(m)h(n - m)W_N^{-(n-m)k} \quad (8.7)$$

$$= W_N^{-nk} \sum_{m=-\infty}^{\infty} x(m)W_N^{mk}h(n - m) = W_N^{-nk}X(n, k). \quad (8.8)$$

From (8.6) and (8.8) it is important to notice that

$$\tilde{X}(n, k) = W_N^{-nk}X(n, k) = W_N^{-nk}|X(n, k)|e^{j\varphi(n, k)} \quad (8.9)$$

$$\tilde{\varphi}(n, k) = \frac{2\pi}{N}n + \varphi(n, k). \quad (8.10)$$

Based on equations (8.7) and (8.8) two different implementations are possible, as shown in Fig. 8.4. The first implementation is the so-called complex baseband implementation according to (8.8). The baseband signals $X(n, k)$ (short-time Fourier transform) are computed by modulation of $x(n)$ with W_N^{nk} and lowpass filtering for each channel k . The modulation of $X(n, k)$ by W_N^{-nk} yields the bandpass signal $\tilde{X}(n, k)$. The second implementation is the so-called complex bandpass implementation, which filters the input signal with $h_k(n)$ given by (8.4), as shown in the lower

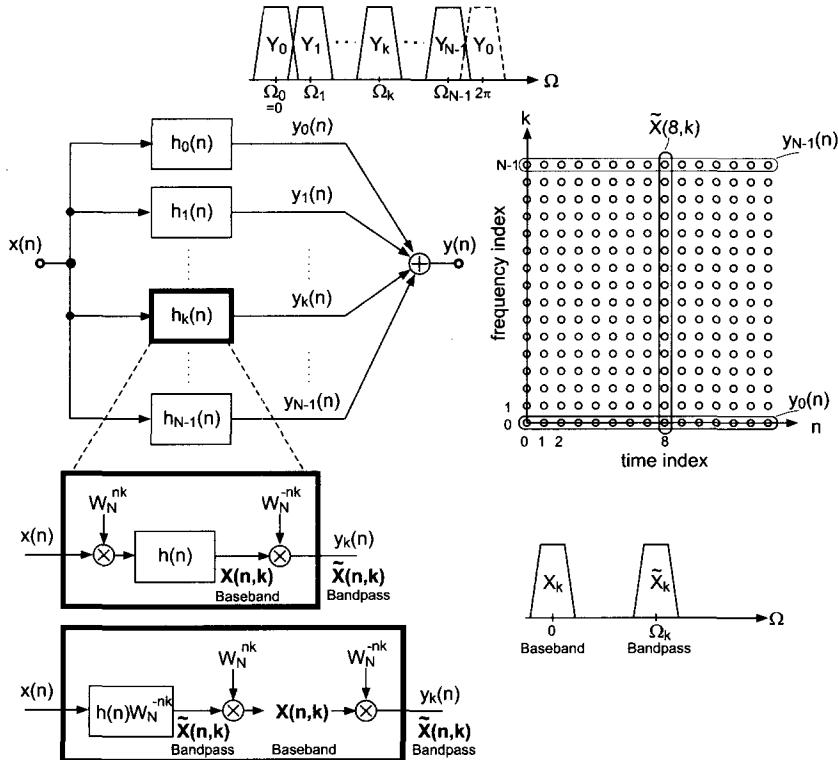


Figure 8.4 Filter bank description of the short-time Fourier transform. Two implementations of the k th channel are shown in the lower left part. The discrete-time and discrete-frequency plane is shown in the right part. The marked bandpass signals $y_k(n)$ are the horizontal samples $\tilde{X}(n, k)$. The different frequency bands Y_k corresponding to each bandpass signal are shown on top of the filter bank. The frequency bands for the baseband signal $X(n, k)$ and the bandpass signal $\tilde{X}(n, k)$ are shown in the lower right part.

left part of Fig. 8.4. This implementation leads directly to the complex-valued bandpass signals $\tilde{X}(n, k)$. If the equivalent baseband signals $X(n, k)$ are necessary, they can be computed by multiplication with W_N^{nk} . The operations for the modulation by W_N^{nk} yielding $X(n, k)$ and back modulation by W_N^{-nk} (lower left part of Fig. 8.4) are only shown to point out the equivalence of both implementations.

The output sequence $y(n)$ is the sum of the bandpass signals according to

$$y(n) = \sum_{k=0}^{N-1} y_k(n) = \sum_{k=0}^{N-1} \tilde{X}(n, k) = \sum_{k=0}^{N-1} X(n, k) W_N^{-nk}. \quad (8.11)$$

The output signals $y_k(n)$ are complex-valued sequences $\tilde{X}(n, k)$. For a real-valued input signal $x(n)$ the bandpass signals satisfy the property $y_k(n) = \tilde{X}(n, k) = \tilde{X}^*(n, N-k) = y_{N-k}^*(n)$. For a channel stacking with $\Omega_k = \frac{2\pi k}{N}$ we get the frequency

bands shown in the upper part of Fig. 8.4. The property $\tilde{X}(n, k) = \tilde{X}^*(n, N - k)$ together with the channel stacking can be used for the formulation of real-valued bandpass signals (real-valued k th channel)

$$\hat{y}_k(n) = \tilde{X}(n, k) + \tilde{X}(n, N - k) = \tilde{X}(n, k) + \tilde{X}^*(n, k) \quad (8.12)$$

$$= |X(n, k)| \cdot [e^{j\tilde{\varphi}(n, k)} + e^{-j\tilde{\varphi}(n, k)}] \quad (8.13)$$

$$= 2|X(n, k)| \cdot \cos[\tilde{\varphi}(n, k)] \quad (8.14)$$

for $k = 1, \dots, N/2 - 1$.

This leads to

$$\hat{y}_0(n) = y_0(n), \quad k = 0 \quad (8.15)$$

dc channel

$$\hat{y}_k(n) = \underbrace{2|X(n, k)|}_{A(n, k)} \cdot \underbrace{\cos[\Omega_k n + \varphi(n, k)]}_{\tilde{\varphi}(n, k)}, \quad k = 1, \dots, N/2 - 1 \quad (8.16)$$

bandpass channels

$$\hat{y}_{N/2}(n) = y_{N/2}(n), \quad k = N/2 \quad (8.17)$$

highpass channel.

Besides a dc and a highpass channel we have $N/2 - 1$ cosine signals with fixed frequencies Ω_k and time-varying amplitude and phase. This means that we can add real-valued output signals $\hat{y}_k(n)$ to yield the output signal

$$y(n) = \sum_{k=0}^{N/2} \hat{y}_k(n). \quad (8.18)$$

This interpretation offers analysis of a signal by a filter bank, modification of the short-time spectrum $\tilde{X}(n, k)$ on a sample-by-sample basis and synthesis by a summation of the bandpass signals $y_k(n)$. Due to the fact that the baseband signals are bandlimited by the lowpass filter $h(n)$, a sampling rate reduction can be performed in each channel to yield $X(sR, k)$, where only every R th sample is taken and s denotes the new time index. This leads to a short-time transform $X(sR, k)$ with a hop size of R samples. Before the synthesis upsampling and interpolation filtering has to be performed [CR83].

8.2.2 Block-by-Block Analysis/Synthesis Model

A detailed description of a phase vocoder implementation using the FFT can be found in [Por76, Cro80, CR83]. The analysis and synthesis implementations are precisely described in [CR83, p. 318, Fig. 7.19 and p. 321, Fig. 7.20]. A simplified analysis and synthesis implementation, where the window length is less or equal to the FFT length, were proposed in [Cro80]. The analysis and synthesis algorithm and the discrete-time and discrete-frequency plane are shown in Fig. 8.5. The analysis

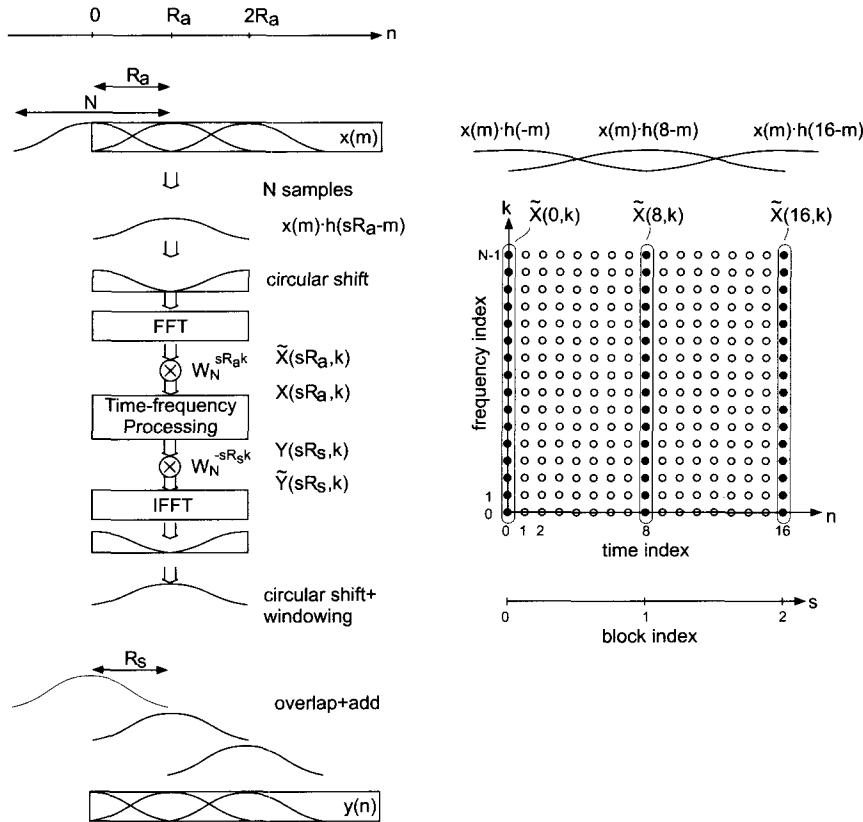


Figure 8.5 Phase vocoder using the FFT/IFFT for the short-time Fourier transform. The analysis hop size R_a determines the sampling of the two-dimensional time-frequency grid. Time-frequency processing allows the reconstruction with a synthesis hop size R_s .

algorithm [Cro80] is given by

$$X(sR_a, k) = \sum_{m=-\infty}^{\infty} x(m)h(sR_a - m)W_N^{mk} \quad (8.19)$$

$$= W_N^{sR_a k} \sum_{m=-\infty}^{\infty} x(m)h(sR_a - m)W_N^{-(sR_a - m)k} \quad (8.20)$$

$$= W_N^{sR_a k} \cdot \tilde{X}(sR_a, k) \quad (8.21)$$

$$= X_R(sR_a, k) + jX_I(sR_a, k) = |X(sR_a, k)| \cdot e^{j\varphi(sR_a, k)} \quad (8.22)$$

$$k = 0, 1, \dots, N - 1,$$

where the short-time Fourier transform is sampled every R_a samples in time and s denotes the time index of the short-time transform at the decimated sampling rate. This means that the time index is now $n = sR_a$, where R_a denotes the analysis hop

size. The analysis window is denoted by $h(n)$. Notice that $X(n, k)$ and $\tilde{X}(n, k)$ in the FFT implementation can also be found in the filter bank approach. The circular shift of the windowed segment before the FFT and after the IFFT is derived in [CR83] and provides a zero-phase analysis and synthesis regarding the center of the window. Further details will be discussed in the next section. Spectral modifications in the time-frequency plane can now be done which yields $Y(sR_s, k)$, where R_s is the synthesis hop size. The synthesis algorithm [Cro80] is given by

$$\begin{aligned} y(n) &= \sum_{s=-\infty}^{\infty} f(n - sR_s) y_s(n - sR_s) \\ \text{with } y_s(n) &= \frac{1}{N} \sum_{k=0}^{N-1} \left[W_N^{-sR_s k} Y(sR_s, k) \right] W_N^{-nk}, \end{aligned} \quad (8.23)$$

where $f(n)$ denotes the synthesis window. Finite length signals $y_s(n)$ are derived from inverse transforms of short-time spectra $Y(sR_s, k)$. These short-time segments are weighted by the synthesis window $f(n)$ and then added by the overlap-add procedure given by (8.23) (see Fig. 8.5).

8.3 Phase Vocoder Implementations

This section describes several phase vocoder implementations for digital audio effects. A useful representation is the time-frequency plane where one displays the values of the magnitude $|X(n, k)|$ and phase $\tilde{\varphi}(n, k)$ of the $\tilde{X}(n, k)$ signal. If the sliding Fourier transform is used as an analysis scheme, this graphical representation is the combination of the spectrogram, which displays the magnitude values of this representation, and the *phasogram* which displays the phase. However, phasograms are harder to read when the hop size is not small. Figure 8.6 shows a spectrogram and a phasogram which correspond to the discrete-time and discrete-frequency plane achieved by a filter bank (see Fig. 8.4) or a block-by-block FFT analysis (see Fig. 8.5) described in the previous section. In a horizontal direction a line represents the output magnitude $|X(n, k)|$ and the phase $\tilde{\varphi}(n, k)$ of the k th analysis bandpass filter over the time index n . In the vertical direction a line represents the magnitude $|X(n, k)|$ and phase $\tilde{\varphi}(n, k)$ for a fixed time index n , which corresponds to a short-time spectrum over frequency bin k at the center of the analysis window located at time index n . The spectrogram in Fig. 8.6 with frequency range up to 2 kHz shows five horizontal rays over the time axis indicating the magnitude of the harmonics of the analyzed sound segment. The phasogram shows the corresponding phases for all five horizontal rays $\tilde{\varphi}(n, k)$, which rotate according to the frequencies of the five harmonics. With a hop size of one we get a visible tree structure. For a larger hop size we get a sampled version, where the tree structure usually disappears.

The analysis and synthesis part can come from the filter bank summation model (see basics), in which case the resynthesis part consists in summing sinusoids, whose amplitudes and frequencies are coming from a parallel bank of filters. The analysis

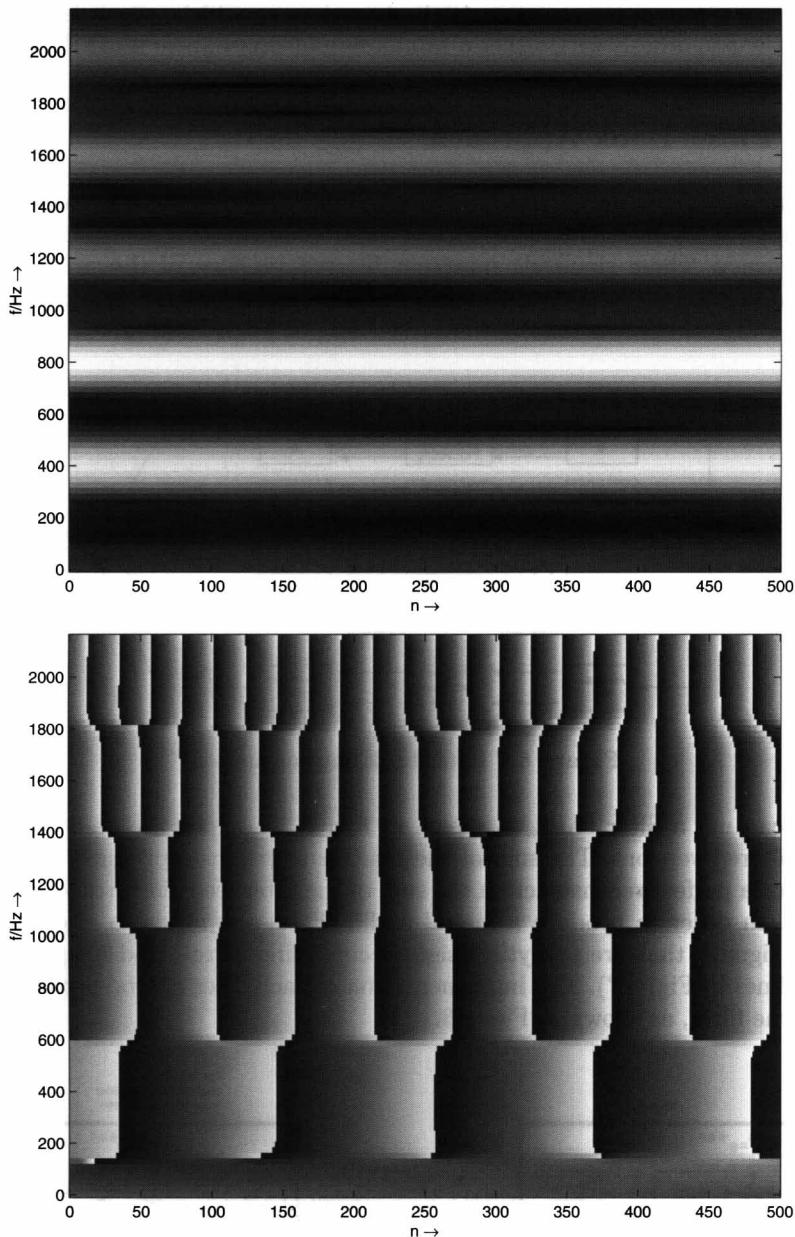


Figure 8.6 Magnitude $|X(n, k)|$ (upper plot) and phase $\tilde{\varphi}(n, k)$ (lower plot) display of a sliding Fourier transform with a hop size $R_a = 1$ or a filter bank analysis approach. For the upper display the grey value (black = 0 and white = maximum amplitude) represents the magnitude range. In the lower display the phase values are in the range $-\pi \leq \tilde{\varphi}(n, k) \leq \pi$.

part can also come from a sliding FFT algorithm, in which case it is possible to perform the resynthesis with either a summation of sinusoids or an IFFT approach.

8.3.1 Filter Bank Approach

From a musician's point of view the idea behind this technique is to represent a sound signal as a sum of sinusoids. Each of these sinusoids is modulated in amplitude and frequency. These sinusoids represent filtered versions of the original signal. The manipulation of the amplitudes and frequencies of these individual signals will produce a digital effect including time stretching or pitch shifting.

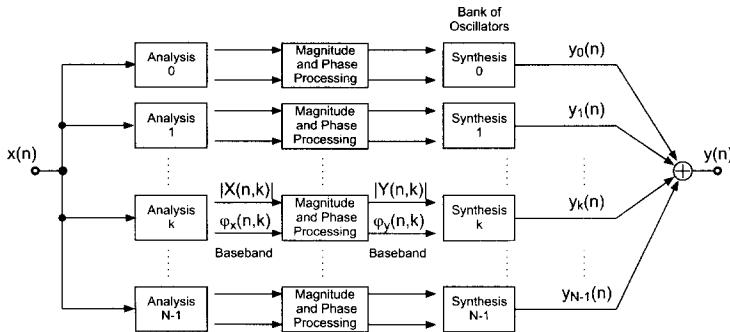


Figure 8.7 Filter bank implementation.

One can use a filter bank, as shown in Fig. 8.7, to split the audio signal into several filtered versions. The sum of these filtered versions reproduces the original signal. For a perfect reconstruction the sum of the filter frequency responses should be unity. In order to produce a digital audio effect, one needs to alter the intermediate signals, that are analytical signals consisting of real and imaginary parts (double lines in Fig. 8.7). The implementation of each filter can be performed by a heterodyne filter, as shown in Fig. 8.8.

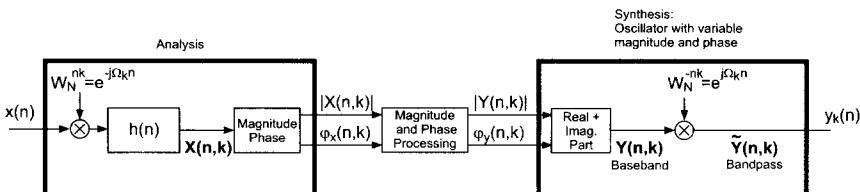


Figure 8.8 Heterodyne filter implementation.

The implementation of a stage of a heterodyne filter consists of a complex-valued oscillator with a fixed frequency Ω_k , a multiplier and an FIR filter. The multiplication shifts the spectrum of the sound, and the FIR filter limits the width

of the frequency shifted spectrum. This heterodyne filtering can be used to obtain intermediate analytic signals, which can be put in the form

$$X(n, k) = [x(n) \cdot e^{-j\Omega_k n}] * h(n) = X_R(n, k) + jX_I(n, k) \quad (8.24)$$

$$= |X(n, k)|e^{j\varphi(n, k)} \quad (8.25)$$

$$X_R(n, k) = |X(n, k)| \cos(\varphi(n, k)) \quad (8.26)$$

$$X_I(n, k) = |X(n, k)| \sin(\varphi(n, k)). \quad (8.27)$$

The difference from classical bandpass filtering is that here the output signal is located in the baseband. This representation leads to a slowly varying phase $\varphi(n, k)$ and the derivation of the phase is a measure of the frequency deviation from the center frequency Ω_k . A sinusoid $x(n) = \cos[\Omega_k n + \varphi_0]$ with frequency Ω_k can be written as $x(n) = \cos[\tilde{\varphi}(n)]$, where $\tilde{\varphi}(n) = \Omega_k n + \varphi_0$. The derivation of $\tilde{\varphi}(n)$ gives the frequency $\Omega_k = \frac{d\tilde{\varphi}(n)}{dn}$. The derivation of the phase $\tilde{\varphi}(n, k)$ at the output of a bandpass filter is termed the *instantaneous frequency* given by

$$\Omega_i(n, k) = \omega_i(n, k)T = 2\pi f_i(n, k)/f_S \quad (8.28)$$

$$= \frac{d}{dn} \tilde{\varphi}(n, k) \quad (8.29)$$

$$= \Omega_k + \frac{d}{dn} \varphi(n, k) \quad (8.30)$$

$$= \Omega_k + \varphi(n, k) - \varphi(n-1, k) \quad (8.31)$$

$$\Rightarrow f_i(n, k) = \left(\frac{k}{N} + \frac{\varphi(n, k) - \varphi(n-1, k)}{2\pi} \right) \cdot f_S. \quad (8.32)$$

The instantaneous frequency can be described in a musical way as the frequency of the filter output signal in the filter bank approach. The phase of the baseband output signal is $\varphi(n, k)$ and the phase of the bandpass output signal is $\tilde{\varphi}(n, k) = \Omega_k n + \varphi(n, k)$ (see Fig. 8.8). As soon as we have the instantaneous frequencies, we can build an oscillator bank and eventually change the amplitudes and frequencies of this bank to build a digital audio effect. The recalculation of the phase from a modified instantaneous frequency is done by computing the phase according to

$$\tilde{\varphi}(n, k) = \tilde{\varphi}(0, k) + \int_0^{nT} 2\pi f_i(\tau, k) d\tau. \quad (8.33)$$

The result of the magnitude and phase processing can be written as $Y(n, k) = |Y(n, k)|e^{j\varphi_y(n, k)}$, which is then used as the magnitude and phase for the complex-valued oscillator running with frequency Ω_k . The output signal is then given by

$$\tilde{Y}(n, k) = |Y(n, k)|e^{j\varphi_y(n, k)} \cdot e^{j\Omega_k n} \quad (8.34)$$

$$= Y(n, k) \cdot e^{j\Omega_k n} = [Y_R(n, k) + jY_I(n, k)] \cdot e^{j\Omega_k n}. \quad (8.35)$$

The resynthesis of the output signal can then be performed by summing all the

individual back shifted signals (oscillator bank) according to

$$y(n) = \sum_{k=0}^{N-1} \tilde{Y}(n, k) = \sum_{k=0}^{N-1} Y(n, k) \cdot e^{j\Omega_k n} \quad (8.36)$$

$$= \sum_{k=0}^{N/2} A(n, k) \cos [\Omega_k n + \varphi_y(n, k)], \quad (8.37)$$

where (8.37) was already introduced by (8.18). The modification of the phases and frequencies for time stretching and pitch shifting needs further explanation and will be treated in a following subsection.

The following M-file 8.1 shows a filter bank implementation with heterodyne filters as shown in Fig. 8.8 (see also Fig. 8.4).

```
M-file 8.1 (VX_het_nothing.m)
% VX_het_nothing.m
clear; clf
===== this program implements a heterodyne filter bank
===== then filters a sound through the filter bank
===== and reconstructs a sound

----- user data -----
WLen      = 256;
nChannel   = 128; % nb of channels
n1         = 1024; % block size for calculation
                  % (must be a multiple of WLen)
[DAFx_in, FS] = wavread('la.wav');
L          = length(DAFx_in);
DAFx_in    = [DAFx_in; zeros(n1,1)] / max(abs(DAFx_in));

----- window and arrays -----
window    = hanningz(WLen);
DAFx_out = zeros(length(DAFx_in),1);
X         = zeros(n1, nChannel);
z         = zeros(WLen-1, nChannel);

----- initialization of the filters -----
t  = (0:n1-1)';
het = zeros(n1,nChannel);
for k=1:nChannel
wk     = 2*pi*i*(k/WLen);
het(:,k) = exp(wk*(t+WLen/2));
het2(:,k) = exp(-wk*t);
end

%colormap(gray); imagesc(angle(het)); axis('xy'); pause;
```

```

tic
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
p      = 0;
pend = length(DAFx_in) - n1;
while p<pend
p
grain = DAFx_in(p+1:p+n1);
%=====
%---- filtering ----
for k=1:nChannel
[X(:,k), z(:,k)] = filter(window, 1, grain.*het(:,k), z(:,k));
end
X_tilde = X.*het2;
%---- drawing ----
% imagesc(angle(X_tilde)); axis('xy'); drawnow
%---- reconstruction ----
res = real(sum(X_tilde,2));
%=====
DAFx_out(p+1:p+n1) = res;
p = p + n1;
end
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
toc

%---- listening and saving the output ----
DAFx_out = DAFx_out(nChannel+1:nChannel+L) / max(abs(DAFx_out));
soundsc(DAFx_out,FS);
wavwrite(DAFx_out, FS, 'la_het_nothing.wav');

```

M-file 8.2 demonstrates the second filter bank implementation with complex-valued bandpass filters, as shown in Fig. 8.4.

```

M-file 8.2 (VX_filter_nothing.m)
% VX_filter_nothing.m
clear; clf
%==== this program performs a complex-valued filter bank
%==== then filters a sound through the filter bank
%==== and reconstructs a sound

%---- user data ----
WLen      = 256;
nChannel   = 128; % nb of channels
n1        = 1024; % block size for calculation
[DAFx_in, FS] = wavread('la.wav');
L          = length(DAFx_in);

```

```

DAFx_in      = [DAFx_in; zeros(n1,1)] / max(abs(DAFx_in));

%----- window and arrays -----
window = hanningz(WLen);
DAFx_out = zeros(length(DAFx_in),1);
X_tilde = zeros(n1,nChannel);
z       = zeros(WLen-1,nChannel);

%----- initialisation of the filters -----
t        = (-WLen/2:WLen/2-1)';
ourFilter = zeros(WLen, nChannel);
for k=1:nChannel
wk        = 2*pi*i*(k/WLen);
ourFilter(:,k) = window.*exp(wk*t);
end

%colormap(gray)

tic
%%%%%%%%%%%%%
p    = 0;
pend = length(DAFx_in) - n1;
while p<pend
p
grain = DAFx_in(p+1:p+n1);
%=====
%----- filtering -----
for k=1:nChannel
[X_tilde(:,k),z(:,k)]=filter(ourFilter(:,k),1,grain,z(:,k));
end
%----- drawing -----
% imagesc(angle(X_tilde')); axis('xy'); drawnow;
%----- reconstruction -----
res = real(sum(X_tilde,2));
%=====
DAFx_out(p+1:p+n1) = res;
p = p + n1;
end
%%%%%%%%%%%%%
toc

%----- listening and saving the output -----
DAFx_out = DAFx_out(nChannel+1:nChannel+L) / max(abs(DAFx_out));
soundsc(DAFx_out, FS);
wavwrite(DAFx_out, FS, 'la_filter_nothing.wav');

```

8.3.2 Direct FFT/IFFT Approach

The FFT algorithm also calculates the values of the magnitudes and phases within a time frame allowing a shorter calculation time. So many analysis-synthesis algorithms use this transform. There are different ways to interpret a sliding Fourier transform, and consequently to invent a method of resynthesis starting from this time-frequency representation. The first one is to apply the inverse FFT on each short-time spectrum and use the overlap-add method to reconstruct the signal. The second one is to consider a horizontal line of the time-frequency representation (constant frequency versus time) and to reconstruct a filtered version for each line. The third one is to consider each point of the time-frequency representation and to make a sum of small grains called *gaborets*. In each interpretation one must test the ability of obtaining a perfect reconstruction if one does not modify the representation. Another important fact is the ability to provide effect implementations that do not have too many artifacts when one modifies on purpose the values of the sliding FFT, especially in operations such as time stretching or filtering.

We now describe the direct FFT/IFFT approach. A time-frequency representation can be seen as a series of overlapping FFTs with or without windowing. As the FFT is invertible, one can reconstruct a sound by adding the inverse FFT of a vertical line (constant time versus frequency), as shown in Fig. 8.9.

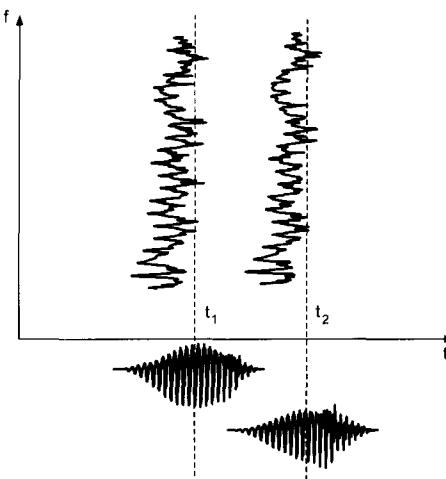


Figure 8.9 FFT and IFFT: vertical line interpretation. At two time instances two spectra are used to compute two time segments.

A perfect reconstruction can be achieved, if the sum of the overlapping windows is unity (see Fig. 8.10). A modification of the FFT values can produce time aliasing, which can be avoided by either zero-padded windows or using windowing after the inverse FFT. In this case the product of the two windows has to be unity. An example is shown in Fig. 8.11. This implementation will be used most frequently in this chapter.

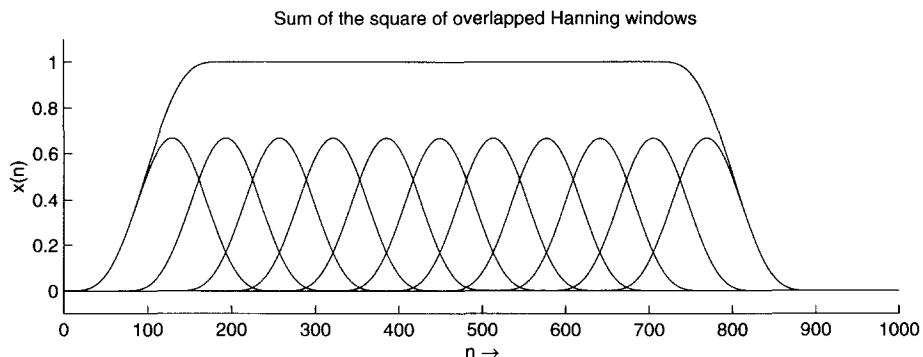


Figure 8.10 Sum of small windows.

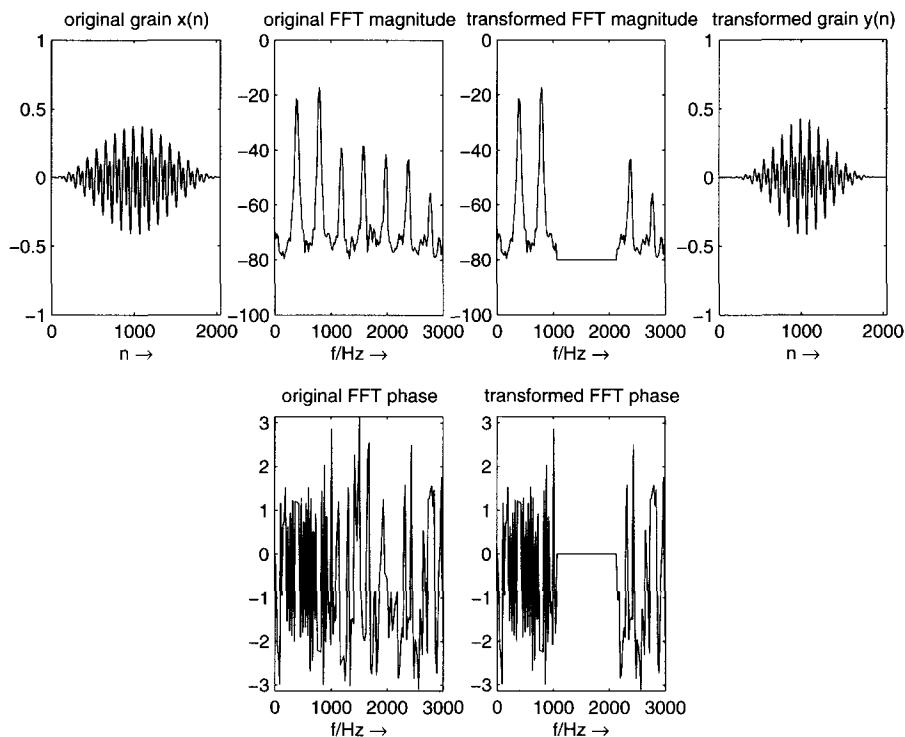


Figure 8.11 Sound windowing, FFT modification and IFFT.

The following M-file 8.3 shows a phase vocoder implementation based on the direct FFT/IFFT approach, where the routine itself is given two vectors for the sound, a window and a hop size.

```
M-file 8.3 (VX_pv_nothing.m)
% VX_pv_nothing.m
%===== this program is a simple phase vocoder, with:
%===== w1 and w2 windows (analysis and synthesis)
%===== WLen is the length of the windows
%===== n1 and n2: steps (in samples) for the analysis and synthesis
clear; clf
%---- user data ----
n1          = 512;
n2          = n1;
WLen        = 2048;
w1          = hanningz(WLen);
w2          = w1;
[DAFx_in, FS] = wavread('la.wav');
L = length(DAFx_in);
DAFx_in     = [zeros(WLen, 1); DAFx_in; ...
    zeros(WLen-mod(L,n1),1)] / max(abs(DAFx_in));
%---- initializations ----
DAFx_out = zeros(length(DAFx_in),1);
tic
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
pin = 0;
pout = 0;
pend = length(DAFx_in) - WLen;

while pin<pend
grain = DAFx_in(pin+1:pin+WLen).* w1;
%=====
f      = fft(fftshift(grain));
r      = abs(f);
phi   = angle(f);
ft    = (r.* exp(i*phi));
grain = fftshift(real(ifft(ft))).*w2;
% =====
DAFx_out(pout+1:pout+WLen) = ...
    DAFx_out(pout+1:pout+WLen) + grain;
pin = pin + n1;
pout = pout + n2;
end
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
toc
%---- listening and saving the output ----
%DAFx_in = DAFx_in(WLen+1:WLen+L);
DAFx_out = DAFx_out(WLen+1:WLen+L) / max(abs(DAFx_out));
soundsc(DAFx_out, FS);
wavwrite(DAFx_out, FS, 'la_pv_nothing.wav');
```

The kernel algorithm performs successive FFTs, inverse FFTs and overlap-add of successive grains. The key point of the implementations is how to go from the FFT representation, where the time origin is at the beginning of the window to the phase vocoder representation used in section 8.2, either in its filter bank description or its block-by-block approach.

The first problem we have to solve is the fact that the time origin for an FFT is on the left of the window. We would like to have it centered, so that for example the FFT of a centered impulse would be zero phase. This is done by a circular shift of the signal, which is a commutation of the first and second part of the buffer. The discrete-time Fourier transform of $\hat{x}(n) = x(n - N/2)$ is $\hat{X}(e^{j\Omega}) = e^{-j\Omega \frac{N}{2}} X(e^{j\Omega})$. With $\Omega_k = \frac{2\pi}{N} k$ the discrete Fourier transform gives $\hat{X}(k) = e^{-j\frac{2\pi}{N} k \frac{N}{2}} X(k)$, which is equivalent to $\hat{X}(k) = (-1)^k X(k)$. The circular shift in time domain can be achieved by multiplying the result of the FFT by $(-1)^k$. With this circular shift, the output of the FFT is equivalent to a filter bank, with zero phase filters. When analyzing a sine wave, the display of the values of the phase $\tilde{\varphi}(n, k)$ of the time-frequency representation will follow the phase of the sinusoid. When analyzing a harmonic sound, one obtains a tree with successive branches corresponding to every harmonic (top of Fig. 8.12).

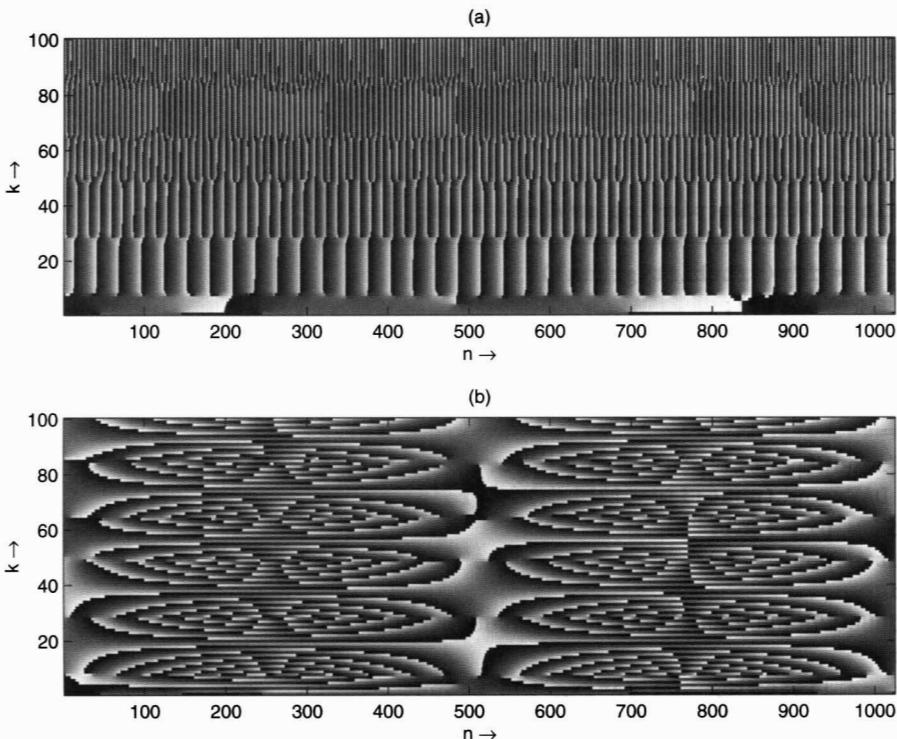


Figure 8.12 Different phase representations: (a) $\tilde{\varphi}(n, k)$ and (b) $\varphi(n, k) = \tilde{\varphi}(n, k) - 2\pi m k / N$.

If we want to take an absolute value as the origin of time, we have to switch to the notation used in section 8.2. We have to multiply the result of the FFT by W_N^{mk} where m is the time sample in the middle of the FFT and k is the number of the bin of the FFT. In this way the display of the phase $\varphi(n, k)$ (bottom of Fig. 8.12) corresponds to a frequency which is the difference between the frequency of the analyzed signal (here a sine wave) delivered by the FFT and the analyzing frequency (the center of the bin). The phase $\varphi(n, k)$ is calculated as $\varphi(n, k) = \tilde{\varphi}(n, k) - 2\pi mk/N$ (N length of FFT, k number of the bin, m time index).

8.3.3 FFT Analysis/Sum of Sinusoids Approach

Conversely, one can read a time-frequency representation with horizontal lines, as shown in Fig. 8.13. Each point on a horizontal line can be seen as the convolution of the original signal with an FIR filter, whose filter coefficients have been given by (8.4). The filter bank approach is very close to the heterodyne filter implementation. The difference comes from the fact that for heterodyne filtering the complex exponential is running with time and the sliding FFT is considering for each point the same phase initiation of the complex exponential. It means that the heterodyne filter measures the phase deviation between a cosine and the filtered signal and the sliding FFT measures the phase with a time origin at zero.

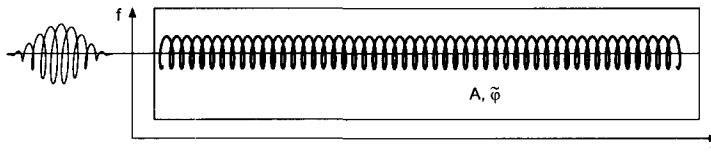


Figure 8.13 Filter bank approach: horizontal line interpretation.

The reconstruction of a sliding FFT on a horizontal line with a hop size of one is performed by filtering of this line with the filter corresponding to the frequency bin (see Fig. 8.13). However, if the analysis hop size is greater than one, we need to interpolate the magnitude values $|X(n, k)|$ and phase values $\tilde{\varphi}(n, k)$. Phase interpolation is based on phase unwrapping, which will be explained in section 8.3.5. Combining phase interpolation with linear interpolation of the magnitudes $|X(n, k)|$ allows the reconstruction of the sound by the addition of a bank of oscillators as given in (8.37).

M-file 8.4 illustrates the interpolation and the sum of sinusoids. Starting from the magnitudes and phases taken from a sliding FFT the synthesis implementation is performed by a bank of oscillators. It uses linear interpolation of the magnitudes and phases.

```
M-file 8.4 (VX_bank_nothing.m)
% VX_bank_nothing.m
%===== this program performs an FFT analysis and
```

```
%===== oscillator bank synthesis, with:  
%===== WLen is the length of the windows  
%===== n1 and n2: steps (in samples) for the analysis and synthesis  
%===== w1 and w2 windows (analysis and synthesis)  
clear; clf  
%---- user data -----  
n1 = 200;  
n2 = n1;  
WLen = 2048;  
w1 = hanningz(WLen);  
w2 = w1;  
[DAFx_in, FS] = wavread('la.wav');  
L = length(DAFx_in);  
DAFx_in = [zeros(WLen, 1); DAFx_in; ...  
zeros(WLen-mod(L,n1),1)] / max(abs(DAFx_in));  
%---- some initializations -----  
DAFx_out = zeros(length(DAFx_in),1);  
ll = WLen/2;  
omega = 2*pi*n1*[0:ll-1]/WLen;  
phi0 = zeros(ll,1);  
r0 = zeros(ll,1);  
psi = zeros(ll,1);  
grain = zeros(WLen,1);  
res = zeros(n2,1);  
  
tic  
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU  
pin = 0;  
pout = 0;  
pend = length(DAFx_in) - WLen;  
  
while pin<pend  
grain = DAFx_in(pin+1:pin+WLen).* w1;  
%=====  
fc = fft(fftshift(grain));  
f = fc(1:ll);  
r = abs(f);  
phi = angle(f);  
delta_phi = omega + princarg(phi-phi0-omega);  
    % now we have the unwrapped difference of phase  
    % on each bin for the hop size of n2  
delta_r = (r-r0)/n1;  
delta_psi = delta_phi/n1;  
    % and now we have the increment of phase and of magnitude  
    % to make a linear interpolation and reconstruction  
for k=1:n2
```

```

r0      = r0 + delta_r;
psi     = psi + delta_psi;
res(k) = r0'*cos(psi);
    % this tricky line is making the sum of weighted cosine
end
%----- for next time -----
phi0 = phi;
r0   = r;
psi  = princarg(psi);
% =====
DAFx_out(pout+1:pout+n2) = DAFx_out(pout+1:pout+n2) + res;
pin  = pin + n1;
pout = pout + n2;
end
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
toc

%----- listening and saving the output -----
%DADF_in = DADF_in(WLen+1:WLen+L);
DAFx_out = DAFx_out(WLen/2+n1+1:WLen/2+n1+L) / max(abs(DAFx_out));
soundsc(DAFx_out,FS);
wavwrite(DAFx_out, FS, 'la_bank_nothing.wav');

```

8.3.4 Gaboret Approach

The idea of the “Gaboret Approach” is the reconstruction of a signal from a time-frequency representation with the sum of “gaborets” weighted by the values of the time-frequency representation [AD93]. The shape of a gaboret is a windowed exponential (see Fig. 8.14), which can be given by $g_{\Omega_k}(n) = e^{-j\Omega_k n} g_\alpha(n)$. The approach is based on the Gabor transform, which is a short-time Fourier transform with the smallest time-frequency window, namely a Gaussian function $g_\alpha(n) = \frac{1}{\sqrt{2\pi}\alpha} e^{-\frac{n^2}{2\alpha}}$ with $\alpha > 0$. The discrete-time Fourier transform of $g_\alpha(n)$ is again a Gaussian function in the Fourier domain. The Gaboret approach is very similar to the wavelet transform [CGT89, Chu92]: one does not consider time or frequency as a privileged axis and one point of the time-frequency plane is the scalar product of the signal with a small gaboret. Further details can be found in [QC93, WR90]. The reconstruction from a time-frequency representation is the sum of gaborets weighted by the values of this time-frequency plane according to

$$y(n) = \sum_{s=-\infty}^{\infty} \sum_{k=0}^{N-1} Y(sR_s, k) f(n - sR_s) W_N^{-nk}. \quad (8.38)$$

Although this point of view is totally equivalent to windowing plus FFT/IFFT plus windowing, it allows a good comprehension of what happens in case of modification of a point in the plane.

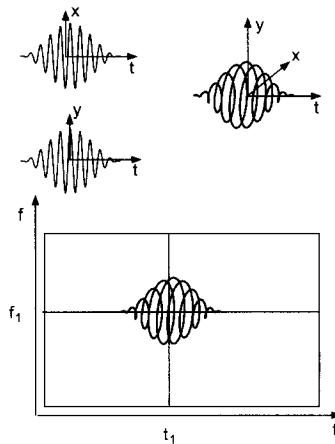


Figure 8.14 Gaboret approach: the upper left part shows real and imaginary values of a gaboret and the upper right part shows a possible 3D representation with axis t , x and y . The lower part shows a gaboret associated to a specific point of a time-frequency representation (for every point we can generate a gaboret in the time domain and then make the sum of all gaborets).

The reconstruction of one single point of a time-frequency representation yields a gaboret in the time domain, as shown in Fig. 8.15. Then a new time-frequency representation of this gaboret is computed. We get a new image, which is called the *reproducing kernel* associated to the transform. This new time-frequency representation is different from the single point of the original time-frequency representation.

So a time-frequency representation of a real signal has some constraints: each value of the time-frequency plane must be the convolution of the neighborhood by the *reproducing kernel* associated to the transformation. This means that if an image (time-frequency representation) is not valid and that we force the reconstruction of a sound by the weighted summation of gaborets, the time-frequency representation of this transformed sound will be in a different form than the initial time-frequency representation. There is no way to avoid this and the beautiful art of making good transforms often relies on the ability to provide “quasi-valid” representations [AD93].

This *reproducing kernel* is only a 2-D extension of the well-known problem of windowing: we find the shape of the FFT of the window around one horizontal ray. But it brings new aspects. When we have two spectral lines, their time-frequency representations are blurred and, when summed, appear as beats. Illustrative examples are shown in Fig. 8.16. The shape of the *reproducing kernel* depends on the shape of the window and is the key point for differences in representations between different windows. The matter of finding spectral lines starting from time-frequency representations is the subject of Chapter 10. Here we only consider the fact that any signal can be generated as the sum of small gaborets. Frequency estimations in bins are obviously biased by the interaction between rays and additional noise.

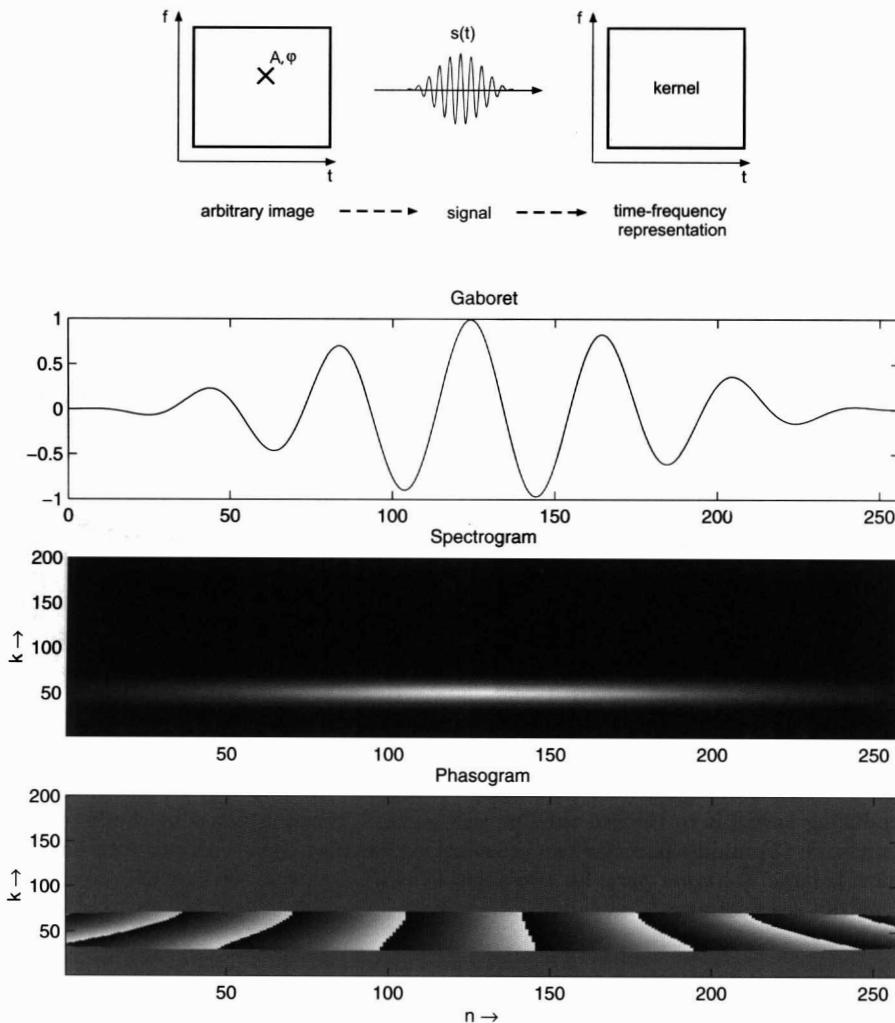


Figure 8.15 Reproducing kernel: the lower three plots represent the forced gaboret and the reproducing kernel consisting of spectrogram and phasogram. (Note: phase values only make sense when the magnitude is not too small.)

The following M-file 8.5 demonstrates the Gaboret analysis and synthesis approach.

```
M-file 8.5 (VX_gab_nothing.m)
% VX_gab_nothing.m
% this program performs the convolution of the signal with gaborets
clear; clf
%---- user data -----
WLen = 512;
```

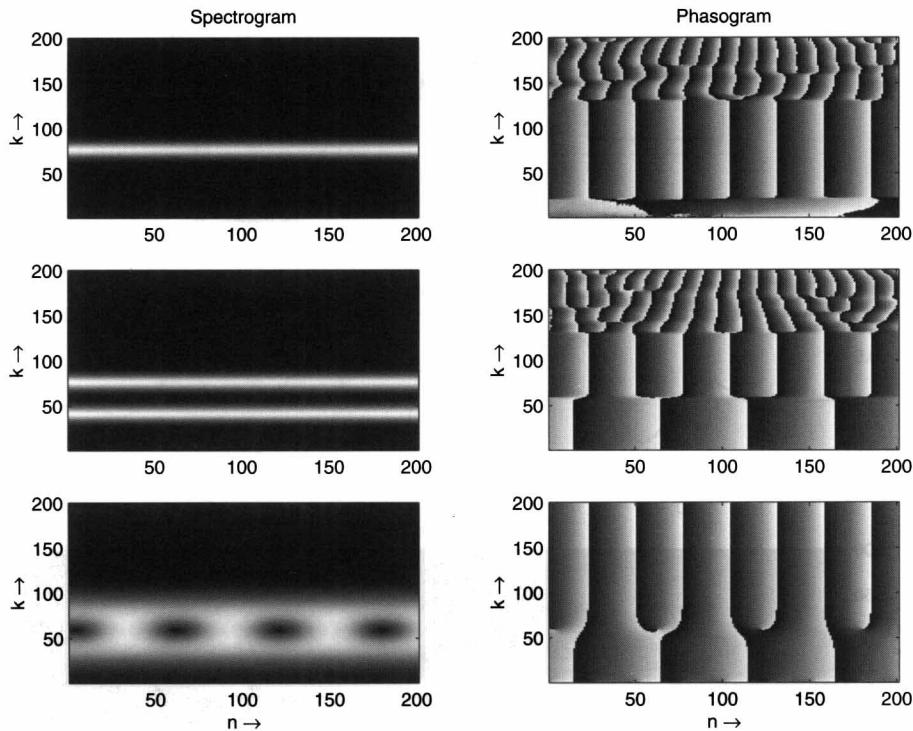


Figure 8.16 Spectrogram and phasogram examples: (a) upper part: the effect of the reproducing kernel is to thicken the line and giving a rotating phase at the frequency of the sinusoid. (b) middle part: for two sinusoids we have two lines with two rotations, if the window is large. (c) lower part: for two sinusoids with a shorter window the two lines mix and we can see beatings.

```

window      = hanningz(WLen);
nChannel    = WLen/2;
n1          = 128;
n2          = n1;
[DAFx_in, FS] = wavread('la.wav');
L           = length(DAFx_in);
DAFx_in     = [zeros(WLen, 1); DAFx_in; ...
              zeros(WLen-mod(L,n1),1)] / max(abs(DAFx_in));

DAFx_out = zeros(length(DAFx_in),1);
%----- initializations calculation of gaborets -----
t   = (-WLen/2:WLen/2-1);
gab = zeros(nChannel,WLen);
for k=1:nChannel
wk      = 2*pi*i*(k/WLen);

```

8.3.5 Phase Unwrapping and Instantaneous Frequency

For the tasks of phase interpolation and instantaneous frequency calculation for every frequency bin k we need a phase unwrapping algorithm. Starting from Fig. 8.4 we perform unwrapping of

$$\tilde{\varphi}(n, k) = \underbrace{\frac{2\pi k}{N}}_{\Omega_k} n + \varphi(n, k)$$

by the unwrapping of $\varphi(n, k)$ and adding the phase variation given by $\Omega_k n$ for all k , as already shown by (8.10). We also need a special function which puts an arbitrary radian phase value into the range $]-\pi, \pi]$. We will call this function *principle argument* [GBA00], which is defined by the expression $y = \text{princarg}[2\pi m + \varphi_x] = \varphi_x$, where $-\pi < \varphi_x \leq \pi$ and m is an integer number. The corresponding Matlab function is shown in Fig. 8.17.

```
function phase=princarg(phasein)
phase=mod(phasein+pi,-2*pi)+pi;
```

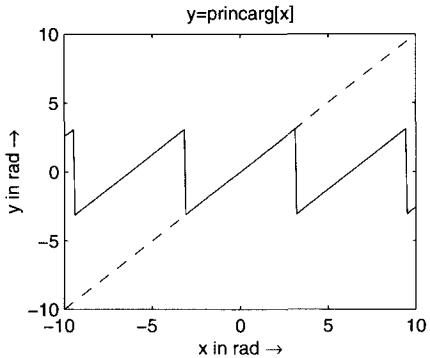


Figure 8.17 Principle argument function (Matlab code and illustrative plot).

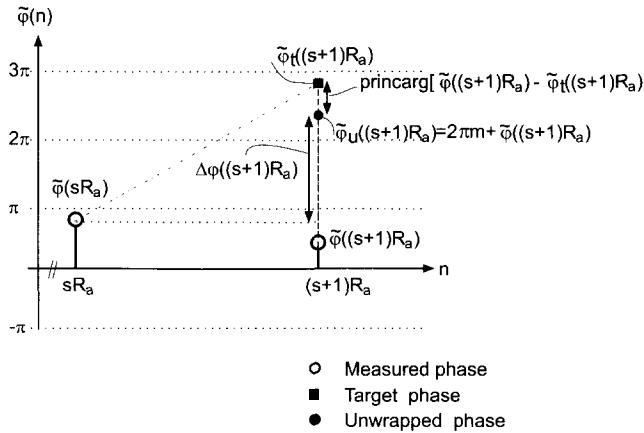


Figure 8.18 Basics of phase computations for frequency bin k .

The phase computations are based on the phase values $\tilde{\varphi}(sR_a, k)$ and $\tilde{\varphi}((s + 1)R_a, k)$, which are the results of the FFTs of two consecutive frames. These phase values are shown in Fig. 8.18. We now consider the phase values regardless of the frequency bin k . If a stable sinusoid with frequency Ω_k exists, we can compute a target phase $\tilde{\varphi}_t((s + 1)R_a)$ from the previous phase value $\tilde{\varphi}(sR_a)$ according to

$$\tilde{\varphi}_t((s + 1)R_a) = \tilde{\varphi}(sR_a) + \Omega_k R_a. \quad (8.39)$$

The unwrapped phase

$$\tilde{\varphi}_u((s + 1)R_a) = \tilde{\varphi}_t((s + 1)R_a) + \tilde{\varphi}_d((s + 1)R_a) \quad (8.40)$$

is computed by the target phase $\tilde{\varphi}_t((s + 1)R_a)$ plus a deviation phase $\tilde{\varphi}_d((s + 1)R_a)$. This deviation phase can be computed by the measured phase $\tilde{\varphi}((s + 1)R_a)$ and the target phase $\tilde{\varphi}_t((s + 1)R_a)$ according to

$$\tilde{\varphi}_d((s + 1)R_a) = \text{princarg}[\tilde{\varphi}((s + 1)R_a) - \tilde{\varphi}_t((s + 1)R_a)]. \quad (8.41)$$

Now we formulate the unwrapped phase (8.40) with the deviation phase (8.41), which leads to the expression

$$\begin{aligned}\tilde{\varphi}_u((s+1)R_a) &= \tilde{\varphi}_t((s+1)R_a) + \text{princarg}[\tilde{\varphi}((s+1)R_a) - \tilde{\varphi}_t((s+1)R_a)] \\ &= \tilde{\varphi}(sR_a) + \Omega_k R_a + \text{princarg}[\tilde{\varphi}((s+1)R_a) - \tilde{\varphi}(sR_a) - \Omega_k R_a].\end{aligned}$$

From the previous equation we can derive the unwrapped phase difference

$$\begin{aligned}\Delta\varphi((s+1)R_a) &= \tilde{\varphi}_u((s+1)R_a) - \tilde{\varphi}(sR_a) \\ &= \Omega_k R_a + \text{princarg}[\tilde{\varphi}((s+1)R_a) - \tilde{\varphi}(sR_a) - \Omega_k R_a]\end{aligned}\quad (8.42)$$

between two consecutive frames. From this unwrapped phase difference we can calculate the instantaneous frequency for frequency bin k at time instant $(s+1)R_a$ by

$$f_i((s+1)R_a) = \frac{1}{2\pi} \frac{\Delta\varphi((s+1)R_a)}{R_a} f_s. \quad (8.43)$$

The Matlab instructions for the computation of the unwrapped phase difference given by (8.42) for every frequency bin k are given here:

```
omega = 2*pi*n1*[0:ll-1]'/WLen;
% ll = N/2 with N length of the FFT
% n1=R_a
delta_phi= omega+princarg(phi-phi0-omega);
```

The term `phi` represents $\tilde{\varphi}((s+1)R_a)$ and `phi0` the previous phase value $\tilde{\varphi}(sR_a)$. In this manner `delta_phi` represents the unwrapped phase variation $\Delta\varphi((s+1)R_a)$ between two successive frames for every frequency bin k .

8.4 Phase Vocoder Effects

The following subsections will describe several modifications of a time-frequency representation before resynthesis in order to achieve audio effects. Most of them use the FFT analysis followed by either a summation of sinusoids or an IFFT synthesis, which is faster or more adapted to the effect. But all implementations give equivalent results and can be used for audio effects.

8.4.1 Time-frequency Filtering

Filtering a sound can be done with recursive (IIR) or nonrecursive (FIR) filters. However, a musician would like to define or even to draw a frequency response which represents the gain for each frequency band. An intuitive way is to use a time-frequency representation and attenuate certain zones, by multiplying the FFT result

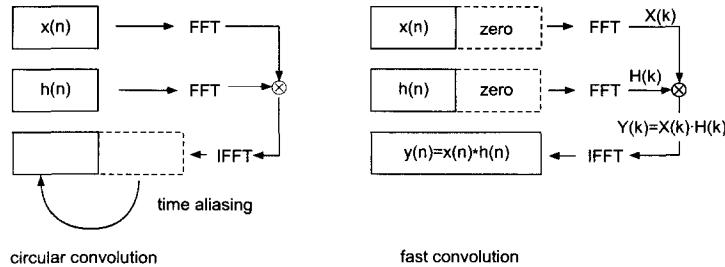


Figure 8.19 Circular convolution and fast convolution.

in every frame by a filtering function in the frequency domain. One must be aware that in that case we are making a circular convolution (during the FFT – inverse FFT process), which leads to time aliasing as shown in Fig. 8.19. The alternative and exact technique for using time-frequency representations is the design of an FIR impulse response from the filtering function. The convolution of the signal segment $x(n)$ of length N with the impulse response of the FIR filter of length $N+1$ leads to an $2N$ -point sequence $y(n) = x(n) * h(n)$. This time domain convolution or filtering can be performed more efficiently in the frequency domain by multiplication of the corresponding FFTs $Y(k) = X(k) \cdot H(k)$. This technique is called *fast convolution* (see Fig. 8.19) and is performed by the following steps:

1. Zero-pad the signal segment $x(n)$ and the impulse response $h(n)$ up to length $2N$.
 2. Take the $2N$ -point FFT of these two signals.
 3. Perform multiplication $Y(k) = X(k) \cdot H(k)$ with $k = 0, 1, \dots, 2N - 1$.
 4. Take the $2N$ -point IFFT of $Y(k)$, which yields $y(n)$ with $n = 0, 1, \dots, 2N - 1$.

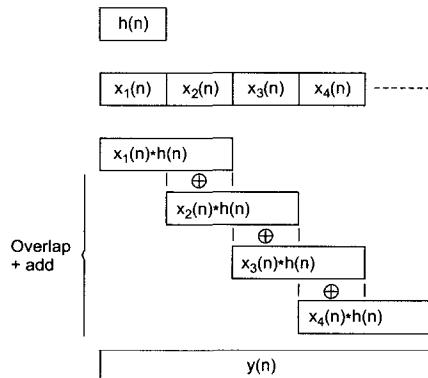


Figure 8.20 FFT filtering.

Now we can work with successive segments of length N of the original signal (which is equivalent to use a rectangular window), zero-pad each segment up to the length $2N$ and perform the fast convolution with the filter impulse response. The results of each convolution are added in an overlap-add procedure, as shown in Fig. 8.20. The algorithm can be summarized as

1. Start from an FIR filter of length $N + 1$, zero pad it to $2N$ and take its FFT $\Rightarrow H(k)$.
2. Partition the signal into segments $x_i(n)$ of length N and zero-pad each segment up to length $2N$.
3. For each zero-padded segment $s_i(n)$ perform the FFT $\Rightarrow X_i(k)$ with $k = 0, 1, \dots, 2N - 1$.
4. Perform the multiplication $Y_i(k) = X_i(k) \cdot H(k)$.
5. Take the inverse FFT of these products $Y_i(k)$.
6. Overlap-add the convolution results (see Fig. 8.20).

The following M-file 8.6 demonstrates the FFT filtering algorithm.

```
M-file 8.6 (UX_filter.m)
% VX_filter.m
%===== this program performs time-frequency filtering
%===== calculation of the fir (here band pass)

clear; clf

%---- user data -----
FirLength      = 1280;          % length of the fir
WLen           = 2*FirLength;   % for zero padding
[DAFx_in, FS] = wavread('la.wav');
L              = length(DAFx_in);
DAFx_in       = [DAFx_in; zeros(WLen-mod(L,FirLength),1)] ...
               / max(abs(DAFx_in));

%---- initializations -----
x              = 1:FirLength;
fr             = 1000/FS;
alpha          = -0.002;
fir            = (exp(alpha*x).*sin(2*pi*fr*x)); % FIR coefficients
plot(fir);

fir2           = [fir; zeros(WLen-FirLength,1)];
fcorr          = fft(fir2);
DAFx_out       = zeros(length(DAFx_in)+FirLength,1);
```

```

grain      = zeros(WLen,1);

tic
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
pin  = 0;
pout = 0;
pend = length(DAFx_in) - FirLength;

while pin<pend
grain = [DAFx_in(pin+1:pin+FirLength); zeros(FirLength,1)];
%=====
ft    = fft(grain) .* fcorr;
grain = (real(ifft(ft)));
%=====
DAFx_out(pin+1:pin+WLen) = ...
    DAFx_out(pin+1:pin+WLen) + grain;
pin   = pin + FirLength;
end
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
toc

----- listening and saving the output -----
%DAFx_in  = DAFx_in(WLen+1:WLen+L);
DAFx_out = DAFx_out / max(abs(DAFx_out));
soundsc(DAFx_out, FS);
wavwrite(DAFx_out, FS, 'la_filter.wav');

```

The design of an N -point FIR filter derived from frequency domain specifications is a classical problem of signal processing. A simple design algorithm is the frequency sampling method [Zöl97].

8.4.2 Dispersion

When a sound is transmitted over telecommunications lines, some of the frequency bands are delayed. This spreads a sound in time, with some components of the signal being delayed. It is usually considered a default in telecommunications but can be used musically. This dispersion effect is especially significant on transients, where the sound loses its coherence, but can also *blur* the steady state parts.

A dispersion effect can be simulated by a filter, especially an FIR filter, whose frequency response has a frequency-dependent time delay. The only change to the previous program is to change the calculation of the FIR vector `fir`. We will now describe several filter designs for a dispersion effect.

Design 1. As an example, a linear chirp signal is a sine wave with linearly increasing frequency and has the property of having a time delay proportional to its frequency. A mathematical definition of a linear chirp signal starting from frequency

zero and going to frequency f_1 during time t_1 is given by

$$\text{Chirp}(t) = \sin(\alpha t^2) \text{ with } \alpha = \pi \frac{f_1}{t_1}. \quad (8.44)$$

Sampling of this chirp signal yields the coefficients for an FIR filter. Time-frequency representations of a linear and an exponential chirp signal are shown in Fig. 8.21a.

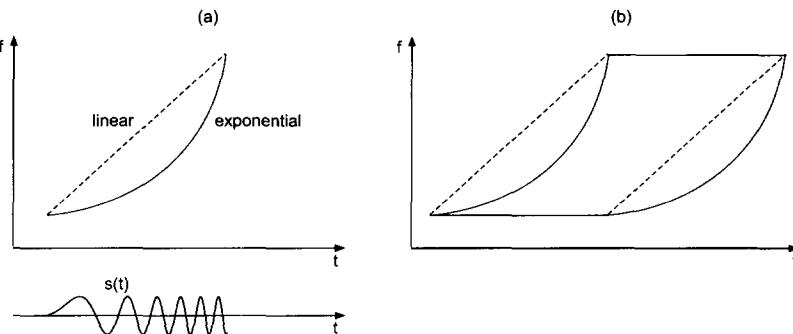


Figure 8.21 Time-frequency representations: (a) linear/exponential chirp signal and (b) time-frequency warping for the linear/exponential chirp.

Design 2. It is also possible to numerically approximate a chirp by integrating an arbitrary frequency function of time. In this case the MATLAB function `cumsum` can be used to calculate the phase $\varphi(n) = \int_0^{nT} 2\pi f(\tau)d\tau + \varphi(0)$ as the integral of the time-dependent frequency $f(t)$. A linear chirp with 300 samples can be computed by the MATLAB instructions:

```
n      = 300;
x      = (1:n)/n;
f0    = 50;
f1    = 4000;
freq  = 2*pi * (f0+(f1-f0)*x) / 44100;
fir   = (sin(cumsum(freq)))';
```

and an exponential chirp by

```
n      = 300;
x      = (1:n)/n;
f0    = 50;
f1    = 4000;
rap   = f1/f0;
freq  = (2*pi*f0/44100) * (rap.^x);
fir   = (sin(cumsum(freq)))';
```

Any other frequency function $f(t)$ can be used for the calculation of `freq`.

Design 3. Nevertheless these chirp signals deliver the frequency as a function of time delay. We would be more likely to define the time delay as a function of frequency. This is only possible with the previous technique if the function is monotonous. Thus in a more general case we can use the phase information of an FFT as an indication of the time delay corresponding to a frequency bin: the phase of a delayed signal $x(n - M)$, which has a discrete Fourier transform $X(k)e^{-jM\frac{2\pi k}{N}}$ with $k = 0, 1, \dots, N/2$, is $\varphi(k) = -M\frac{2\pi k}{N}$ where M is the delay in samples, k is the number of the frequency bin and N is the length of the FFT.

A variable delay for each frequency bin can be achieved by replacing the fixed value M (the delay of each frequency bin) by a function $M(k)$, which leads to $X(k)e^{-jM(k)\frac{2\pi k}{N}}$. For example, a linearly increasing time delay for each frequency bin is given by $M(k) = M \cdot \frac{k}{N}$ with $k = 0, 1, \dots, N/2 - 1$. The derivation of the FIR coefficients can be achieved by performing an IFFT of the positive part of the spectrum and then taking the real part of the resulting complex-valued coefficients. With this technique a linear chirp signal centered around the middle of the window can be computed by the following MATLAB instructions:

```
M      = 300;
WLen = 1024;
mask = [1; 2*ones(WLen/2-1,1); 1 ; zeros(WLen/2-1,1)];
fs   = M*(0:WLen/2)' / WLen; % linear increasing delay
teta = [-2*pi*fs.*((0:WLen/2)'/WLen ; zeros(WLen/2-1,1));
f2   = exp(i*teta);
fir  = fftshift(real(ifft(f2.*mask))));
```

It should be noted that this technique can produce time aliasing. The length of the FIR filter will be greater than M . A proper choice of N is needed, for example $N > 2M$.

Design 4. A final technique is to draw an arbitrary curve on a time-frequency representation, which is an invalid image, and then resynthesize a signal by forcing a reconstruction, for example, by using a summation of gaborets. Then we can use this reconstructed signal as the impulse response of the FIR filter. If the curve displays the dispersion of a filter, we get a dispersive filter.

In conclusion, we can say that dispersion, which is a filtering operation, can be perceived as a delay operation. This leads to a warping of the time-frequency representation, where each horizontal line of this representation is delayed according to the dispersion curve (see Fig. 8.21b).

8.4.3 Time Stretching

Time-frequency scaling is one of the most interesting and difficult tasks that can be assigned to time-frequency representations: changing the time scale independently of the “frequency content”. For example, one can change the rhythm of a song without changing its pitch, or conversely transpose a song without any time change. Time stretching is not a problem that can be stated outside of the perception: we know,

for example, that a sum of two sinusoids is equivalent to a product of a carrier and a modulator. Should a time stretching of this signal still be a sum of two sinusoids or the same carrier with a lower modulation? This leads us to the perception of tremolo tones or vibrato tones. One generally agrees that tremolos and vibratos under 10 Hz are perceived as such and those over are perceived as a sum of sinusoids.

A first technique has been evaluated in the time domain (see PSOLA in section 7.3.3). Here we will deal with another technique in the time-frequency domain using the phase vocoder implementations of section 8.3. There are two implementations for time-frequency scaling by the “traditional” phase vocoder. Historically, the first one uses a bank of oscillators, whose amplitudes and frequencies vary over time. If we can manage to model a sound by the sum of sinusoids, time stretching and pitch shifting can be performed by expanding the amplitude and frequency functions. The second implementation uses the sliding Fourier transform as the model for resynthesis: if we can manage to spread the image of a sliding FFT over time and calculate new phases, then we can reconstruct a new sound with the help of inverse FFTs. Both of these techniques rely on phase interpolation, which need an unwrapping algorithm at the analysis stage, or equivalently an instantaneous frequency calculation, as introduced in section 8.3.5.

The time stretching algorithm mainly consists of providing a synthesis grid which is different from the analysis grid, and to find a way to reconstruct a signal from the values on this grid. Though it is possible to use any stretching factor, we will here only deal with the case where we use an integer both for the analysis hop size R_a , and for the synthesis hop size R_s .

As seen in section 8.3, changing the values and their coordinates on a time-frequency representation is generally not a valid operation, in the sense that the resulting representation is not the sliding Fourier transform of a real signal. However it is always possible to force the reconstruction of a sound from an arbitrary image but the time-frequency representation of the signal issued from this forced synthesis will be different from what was expected. The goal of a good transformation algorithm is to find a strategy that preserves the time stretching aspect without introducing too many artifacts.

The classical way of using a phase vocoder for time stretching is to keep the magnitude unchanged and to modify the phase in such a way that the instantaneous frequencies are preserved. Providing that the grid is enlarged from an analysis hop size R_a to a synthesis hop size R_s , this means that the new phase values must satisfy $\Delta\psi(k) = \frac{R_s}{R_a} \Delta\varphi(k)$ (see Fig. 8.22). Once the grid is filled with these values one can reconstruct a signal using either the filter bank approach or the block-by-block IFFT approach.

Filter Bank Approach (Sum of Sinusoids)

In the FFT analysis/sum of sinusoids synthesis approach, we calculate the instantaneous frequency for each bin and integrate the corresponding phase increment in order to reconstruct a signal as the weighted sum of cosines of the phases. However,

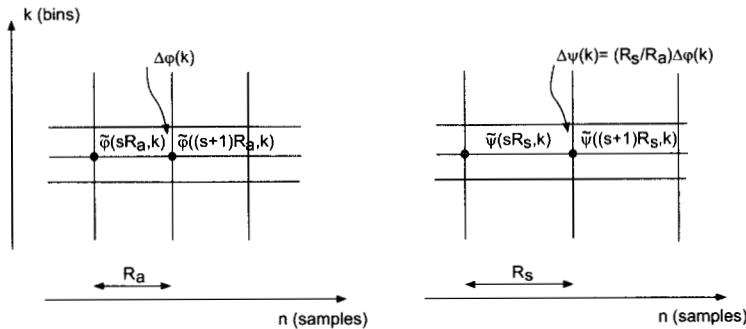


Figure 8.22 Time stretching principle: the analysis with hop size R_a gives the time-frequency grid shown in the left part, where $\Delta\varphi(k) = \tilde{\varphi}((s+1)R_a, k) - \tilde{\varphi}(sR_a, k)$ denotes the phase difference between the unwrapped phases. The synthesis is performed from the modified time-frequency grid with hop size R_s and the phase difference $\Delta\psi(k) = \tilde{\psi}((s+1)R_s, k) - \tilde{\psi}(sR_s, k)$, which is illustrated in the right part.

here the hop size for the resynthesis is different from the analysis. Therefore the following steps are necessary:

1. Calculate the phase increment per sample by $d\psi(k) = \Delta\varphi(k)/R_a$.
2. For the output samples of the resynthesis integrate this value according to $\tilde{\psi}(n+1, k) = \tilde{\psi}(n, k) + d\psi(k)$.
3. Sum the intermediate signals which yields $y(n) = \sum_{k=0}^{N/2} A(n, k) \cos(\tilde{\psi}(n, k))$ (see Fig. 8.23).

A complete MATLAB program for time stretching is given by M-file 8.7.

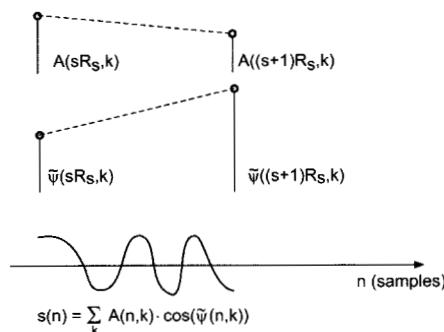


Figure 8.23 Calculation of time-frequency samples. Given the values of A and $\tilde{\psi}$ on the representation grid, we can perform linear interpolation with a hop size of one in between two successive values on the grid. The reconstruction is achieved by a summation of weighted cosines.

```
M-file 8.7 (VX_tstretch_bank.m)
% VX_tstretch_bank.m
%===== this program performs time stretching
%===== using the oscillator bank approach, with:
%===== w1 and w2 windows (analysis and synthesis)
%===== WLen is the length of the windows
%===== n1 and n2: steps (in samples) for the analysis and synthesis

clear; clf

%---- user data ----
n1          = 256;
n2          = 512;
WLen        = 2048;
w1          = hanningz(WLen);
w2          = w1;
[DAFx_in, FS] = wavread('la.wav');
L           = length(DAFx_in);
DAFx_in     = [zeros(WLen, 1); DAFx_in; ...
              zeros(WLen-mod(L,n1),1)] / max(abs(DAFx_in));

%---- initializations ----
tstretch_ratio = n2/n1
DAFx_out = zeros(WLen+ceil(length(DAFx_in)*tstretch_ratio),1);
grain      = zeros(WLen,1);
ll         = WLen/2;
omega      = 2*pi*n1*[0:ll-1]'/WLen;
phi0       = zeros(ll,1);
r0         = zeros(ll,1);
psi         = zeros(ll,1);
res        = zeros(n2,1);

tic
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
pin  = 0;
pout = 0;
pend = length(DAFx_in)-WLen;

while pin<pend
grain = DAFx_in(pin+1:pin+WLen).* w1;
%=====
fc   = fft(fftshift(grain));
f    = fc(1:ll);
r   = abs(f);
phi = angle(f);
delta_phi = omega + princarg(phi-phi0-omega);

```

```

delta_r    = (r-r0)/n2;
delta_psi = delta_phi/n1;
for k=1:n2
r0 = r0 + delta_r;
psi = psi + delta_psi;
res(k) = r0'*cos(psi);
end

phi0 = phi;
r0 = r;
psi = princarg(psi);
% =====
% DAFx_out(pout+1:pout+n2)=DAFx_out(pout+1:pout+n2)+res;
DAFx_out(pout+1:pout+n2) = res;
pin = pin + n1;
pout = pout + n2;
end
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
toc

----- listening and saving the output -----
%DAFx_in = DAFx_in(WLen+1:WLen+L);
DAFx_out=DAFx_out(WLen/2+n1+1:length(DAFx_out))/max(abs(DAFx_out));
soundsc(DAFx_out, FS);
wavwrite(DAFx_out, FS, 'la_tstretch_bank.wav');

```

This program first extracts a series of sound segments called grains. For each grain the FFT is computed to yield a magnitude and phase representation every n_1 samples (n_1 is the analysis hop size R_a). It then calculates a sequence of n_2 samples (n_2 is the synthesis hop size R_s) of the output signal by interpolating the values of r and calculating the phase ψ in such a way that the instantaneous frequency derived from ψ is equal to the one derived from ϕ . The unwrapping of the phase is then done by calculating $(\phi - \phi_0 - \omega)$, putting it in the range $]-\pi, \pi]$ and again adding ω . A phase increment per sample d_{ψ} is calculated from $\delta\phi/n_1$. The calculation of the magnitude and phase at the resynthesis is done in the loop `for k=1:n2` where r and ψ are incremented by d_r and d_{ψ} . The program uses the vector facility of MATLAB to calculate the sum of the cosine of the angles weighted by magnitude in one step. This gives a buffer `res` of n_2 output samples which will be inserted into the `DAFx_out` signal.

Block-by-Block Approach (FFT/IFFT)

Here we follow the FFT/IFFT implementation used in section 8.3, but the hop size for resynthesis is different from the analysis. So we have to calculate new phase values in order to preserve the instantaneous frequencies for each bin. This is again done by calculating an unwrapped phase difference for each frequency bin, which is

proportional to $\frac{R_s}{R_a}$. We also have to take care of some implementation details such as the fact that the period of the window has to be equal to the length of the FFT (this is not the case for the standard MATLAB functions). The synthesis hop size should at least allow a minimal overlap of windows, or should be a submultiple of it. The following M-file 8.8 demonstrates the block-by-block FFT/IFFT implementation.

```
M-file 8.8 (VX_tstretch_real_pv.m)
% VX_tstretch_real_pv.m
%===== this program performs time stretching
%===== using the FFT-IFFT approach,
%===== for real ratio, and using
%===== w1 and w2 windows (analysis and synthesis)
%===== WLen is the length of the windows
%===== n1 and n2: steps (in samples) for the analysis and synthesis

clear;clf

%---- user data -----
n1          = 200;
n2          = 512;
WLen        = 2048;
w1          = hanningz(WLen);
w2          = w1;
[DAFx_in,FS] = wavread('la.wav');
L           = length(DAFx_in);
DAFx_in    = [zeros(WLen, 1); DAFx_in; ...
             zeros(WLen-mod(L,n1),1)] / max(abs(DAFx_in));

%---- initializations -----
tstretch_ratio = n2/n1
DAFx_out = zeros(WLen+ceil(length(DAFx_in)*tstretch_ratio),1);
omega     = 2*pi*n1*[0:WLen-1]/WLen;
phi0      = zeros(WLen,1);
psi       = zeros(WLen,1);

tic
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
pin  = 0;
pout = 0;
pend = length(DAFx_in)-WLen;
while pin<pend
grain = DAFx_in(pin+1:pin+WLen).* w1;
%=====
f      = fft(fftshift(grain));
r      = abs(f);
phi   = angle(f);
```

```

delta_phi= omega + princarg(phi-phi0-omega);
phi0 = phi;
psi = princarg(psi+delta_phi*tstretch_ratio);
ft = (r.* exp(i*psi));
grain = fftshift(real(ifft(ft))).*w2;
% plot(grain);drawnow;
% =====
DAFx_out(pout+1:pout+WLen) = ...
    DAFx_out(pout+1:pout+WLen) + grain;
pin = pin + n1;
pout = pout + n2;
end
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
toc

%---- listening and saving the output ----
%DAFx_in = DAFx_in(WLen+1:WLen+L);
DAFx_out = DAFx_out(WLen+1:length(DAFx_out))/max(abs(DAFx_out));
soundsc(DAFx_out, FS);
wavwrite(DAFx_out, FS, 'la_tstretch_noint_pv.wav');

```

This program is much faster than the preceding one. It extracts grains of the input signal by windowing the original signal `DAFx_in`, makes a transformation of these grains and overlap-adds these transformed grains to get a sound `DAFx_out`. The transformation consists of performing the FFT of the grain and computing the magnitude and phase representation `r` and `phi`. The unwrapping of the phase is then done by calculating $(\phi - \phi_0 - \omega)$, putting it in the range $]-\pi, \pi]$ and again adding `omega`. The calculation of the phase `psi` of the transformed grain is then achieved by adding the phase increment `delta_phi` multiplied by the stretching factor `ral` to the previous unwrapped phase value. As seen before, this is equivalent to keeping the same instantaneous frequency for the synthesis as it is calculated for the analysis. The new output grain is then calculated by an inverse FFT, windowed again and overlap-added to the output signal.

Hints and drawbacks. As we have noticed, phase vocoding can produce artifacts. It is important to know them in order to face them.

1. Changing the phases before the IFFT is equivalent to using an all-pass filter whose Fourier transform contains the phase correction that is being applied. If we do not use a window for the resynthesis, we can ensure the circular convolution aspect of this filtering operation. We will have discontinuities at the edges of the signal buffer. So it is necessary to use a synthesis window.
2. Nevertheless, even with a resynthesis window (also called *tapering window*) the circular aspect still remains: the result is the aliased version of an infinite IFFT. A way to counteract this is to choose a zero-padded window for analysis and synthesis.

3. Shape of the window: one must ensure that a perfect reconstruction is given with a ratio $\frac{R_s}{R_a}$ equal to one (no time stretching). If we use the same window for analysis and synthesis, the sum of the square of the windows, regularly spaced at the resynthesis hop size, should be one.
4. For a Hanning window without zero-padding the hop size R_s has to be a divisor of $N/4$.
5. Hamming and Blackman windows provide smaller side lobes in the Fourier transform. However, they have the inconvenience of being non-zero at the edges so that no tapering is done by using these windows alone. The resynthesis hop size should be a divisor of $N/8$.
6. Truncated Gaussian windows, which are good candidates, provide a sum that always has oscillations, but which can be below the level of perception.

An important problem is the difference of phase unwrapping between different bins, which is not solved by the algorithms we presented: the unwrapping algorithm of the analysis gives a phase that is equal to the measured phase modulo 2π . So the unwrapped phase is equal to the measured phase plus a term that is a multiple of 2π . This second term is not the same for every bin. Because of the multiplication by the time stretching ratio, there is a dispersion of the phases. One cannot even ensure that two identical successive sounds will be treated in the same way. This is in fact the main drawback of the phase vocoder and its removal is still a matter of research [QM98, Fer99, LD99a].

However, when the time stretching ratio is an integer (e.g. time stretching by 200 percent, 300 percent), the unwrapping is no longer necessary in the algorithm, because the 2π modulo relation is still preserved when the phase is multiplied by an integer. The key point here is that we can make a direct multiplication of the analysis phase to get the phase for synthesis. So in this case it is more obvious and elegant to use the following algorithm given by M-file 8.9.

```
M-file 8.9 (VX_tstretch_int_pv.m)
% VX_tstretch_int_pv.m
%===== this program performs time stretching using the phase
%===== vocoder approach, with an integer ratio, with:
%===== w1 and w2 windows (analysis and synthesis)
%===== lfen is the length of the windows
%===== n1 and n2: steps (in samples) for the analysis and synthesis

clear;clf

%---- user data ----
n1          = 64;
n2          = 512;
WLen        = 2048;
w1          = hanningz(WLen);
```

```
w2           = w1;
[DAFx_in,FS] = wavread('la.wav');
L            = length(DAFx_in);
DAFx_in     = [zeros(WLen, 1); DAFx_in; ...
              zeros(WLen-mod(L,n1),1)] / max(abs(DAFx_in));

%----- initializations -----
tstretch_ratio = n2/n1
DAFx_out = zeros(WLen+ceil(length(DAFx_in)*tstretch_ratio),1);
grain    = zeros(WLen,1);

tic
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
pin  = 0;
pout = 0;
pend = length(DAFx_in)-WLen;

while pin<pend
    grain = DAFx_in(pin+1:pin+WLen).* w1;
%=====
    f      = fft(fftshift(grain));
    r      = abs(f);
    phi   = angle(f);
    ft    = (r.* exp(i*tstretch_ratio*phi));
    grain = fftshift(real(ifft(ft))).*w2;
% =====
    DAFx_out(pout+1:pout+WLen) = ...
    DAFx_out(pout+1:pout+WLen) + grain;
    pin   = pin + n1;
    pout  = pout + n2;
end
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
toc

%----- listening and saving the output -----
%DAFx_in  = DAFx_in(WLen+1:WLen+L);
DAFx_out = DAFx_out(WLen+1:length(DAFx_out))/max(abs(DAFx_out));
soundsc(DAFx_out, FS);
wavwrite(DAFx_out, FS, 'la_stretch_int_pv.wav');
```

8.4.4 Pitch Shifting

Pitch shifting is different from frequency shifting: a frequency shift is an addition to every frequency, while pitch shifting is the multiplication of every frequency by a transposition factor. Pitch shifting can be directly linked to time stretching. Resam-

pling a time-stretched signal with the inverse of the time stretching ratio performs pitch shifting and going back to the initial duration of the signal (see Fig. 8.24). There are, however, alternative solutions which allow the direct calculation of a pitch shifted version of a sound.

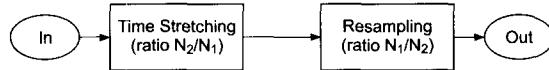


Figure 8.24 Resampling of a time stretching algorithm.

Filter Bank Approach (Sum of Sinusoids)

In the time stretching algorithm using the sum of sinusoids (see section 8.3) we have an evaluation of instantaneous frequencies. As a matter of fact transposing all the instantaneous frequencies can lead to an efficient pitch shifting algorithm. Therefore the following steps have to be performed (see Fig. 8.25):

1. Calculate the phase increment per sample by $d\varphi(k) = \Delta\varphi(k)/R_a$.
2. Multiply the phase increment by the transposition factor `transpo` and integrate the modified phase increment according to $\tilde{\psi}(n+1, k) = \tilde{\psi}(n, k) + \text{transpo} \cdot \Delta\varphi(k)/R_a$.
3. Calculate the sum of sinusoids: when the transposition factor is greater than one, keep only frequencies under the Nyquist frequency bin $N/2$. This can be done by taking only the $N/(2*\text{transpo})$ frequency bins.

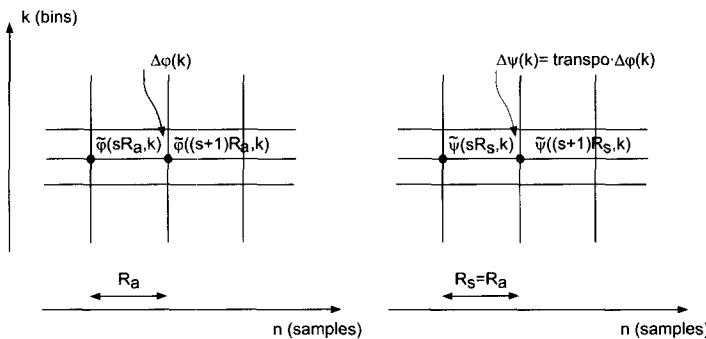


Figure 8.25 Pitch shifting with the filter bank approach: the analysis gives the time-frequency grid with analysis hop size R_a . For the synthesis the hop size is set to $R_s = R_a$ and the phase difference is calculated according to $\Delta\psi(k) = \text{transpo} \cdot \Delta\varphi(k)$.

The following M-file 8.10 is similar to the program given by M-file 8.7 with the exception of a few lines: the definition of the hop size and the resynthesis phase increment have been changed.

```

M-file 8.10 (VX_pitch_bank.m)
% VX_pitch_bank.m
%===== this program performs pitch shifting
%===== using the oscillator bank approach, with:
%===== w1 and w2: windows (analysis and synthesis)
%===== WLen: is the length of the windows
%===== n1: step (in samples) for the analysis and synthesis
%===== pit_ratio: pitch shifting ratio

clear; clf

%---- user data ----
n1      = 512;
pit_ratio = 1.0
WLen    = 2048;
w1      = hanningz(WLen);
w2      = w1;
[DAFx_in, FS] = wavread('la.wav');
L        = length(DAFx_in);
DAFx_in = [zeros(WLen, 1); DAFx_in; ...
           zeros(WLen-mod(L,n1),1)] / max(abs(DAFx_in));

%---- initializations ----
DAFx_out = zeros(length(DAFx_in),1);
grain    = zeros(WLen,1);
ll       = WLen/2;
omega   = 2*pi*n1*[0:ll-1]/WLen;
phi0    = zeros(ll,1);
r0      = zeros(ll,1);
psi     = phi0;
res     = zeros(n1,1);

tic
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
pin  = 0;
pout = 0;
pend = length(DAFx_in)-WLen;

while pin<pend
grain = DAFx_in(pin+1:pin+WLen).* w1;
%=====
fc   = fft(fftshift(grain));
f    = fc(1:ll);
r    = abs(f);
phi = angle(f);
delta_phi = omega + princarg(phi-phi0-omega);

```

```

delta_r    = (r-r0)/n1;
delta_psi = pit_ratio*delta_phi/n1;
for k=1:n1
r0      = r0+delta_r;
psi     = psi+delta_psi;
res(k) = r0'*cos(psi);
end
% plot(res);pause;
phi0 = phi;
r0   = r;
psi  = princarg(psi);
% =====
DAFx_out(pout+1:pout+n1) = DAFx_out(pout+1:pout+n1) + res;
pin  = pin + n1;
pout = pout + n1;
end
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
toc

%---- listening and saving the output ----
%DAFx_in = DAFx_in(WLen+1:WLen+L);
DAFx_out = DAFx_out(WLen/2+n1+1:WLen/2+n1+L) / max(abs(DAFx_out));
soundsc(DAFx_out, FS);
wavwrite(DAFx_out, FS, 'la_pitch_bank.wav');

```

The program is derived from the time-stretching program using the oscillator bank approach in a straightforward way: this time the hop size for analysis and synthesis are the same, and a pitch transpose argument *pit* must be defined. This argument will be multiplied by the phase increment *delta_phi/n1* derived from the analysis to get the phase increment *d_psi* in the calculation loop. This means of course that we consider the pitch transposition as fixed in this program, but easy changes may be done to make it vary with time.

Block-by-Block Approach (FFT/IFFT)

The regular way to deal with pitch shifting using this technique is first to resample the whole output once computed, but this can alternatively be done by resampling the result of every IFFT and overlapping with a hop size equal to the analysis one (see Fig. 8.26). Providing that R_s is a divider of N (FFT length), which is quite a natural way for time stretching (to ensure that the sum of the square of windows is equal to one), one can resample each IFFT result to a length of $N \frac{R_a}{R_s}$ and overlap with a hop size of R_a . Another method of resampling is to use the property of the inverse FFT: if $R_a < R_s$, we can take an IFFT of length $N \frac{R_a}{R_s}$ by taking only the first bins of the initial FFT. If $R_a > R_s$, we can zero pad the FFT, before the IFFT is performed. In each of these cases the result is a resampled grain of length $N \frac{R_a}{R_s}$.

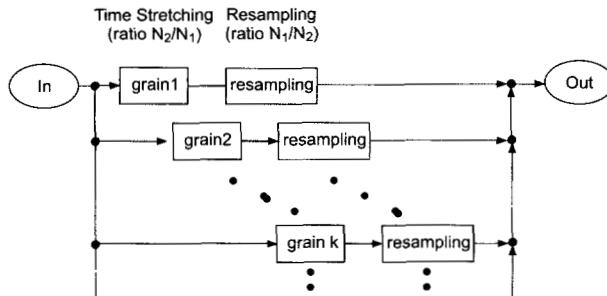


Figure 8.26 Pitch shifting with integrated resampling: for each grain a time stretching and resampling are performed. An overlap-add procedure delivers the output signal.

The following M-file 8.11 implements pitch shifting with integrated resampling according to Fig. 8.26. The M-file is similar to the program given by M-file 8.9, except for the definition of the hop sizes and the calculation for the interpolation.

M-file 8.11 (VX_pitch_pv.m)

```
% VX_pitch_pv.m
%===== this program performs pitch shifting
%===== using the FFT/IFFT approach
%===== w1 and w2: windows (analysis and synthesis)
%===== WLen: is the length of the windows
%===== n1 and n2: steps (in samples) for the analysis and synthesis

clear; clf

n1          = 500;
n2          = 512;
tstretch_ratio = n2/n1;
WLen        = 2048;
w1          = hanningz(WLen);
w2          = w1;
[DAFx_in, FS] = wavread('flute2');
L           = length(DAFx_in);
DAFx_in     = [zeros(WLen, 1); DAFx_in; ...
              zeros(WLen-mod(L,n1),1)] / max(abs(DAFx_in));

%---- for linear interpolation of a grain of length WLen -----
lx = floor(WLen*n1/n2);
x  = 1+(0:lx-1)*WLen/lx;
ix = floor(x);
ix1 = ix+1;
dx = x-ix;
```

```
dx1 = 1-dx;

%----- initializations -----
DAFx_out = zeros(lx+length(DAFx_in),1);
omega    = 2*pi*n1*[0:WLen-1]'/WLen;
phi0     = zeros(WLen,1);
psi      = zeros(WLen,1);

tic
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
pin  = 0;
pout = 0;
pend = length(DAFx_in)-WLen;

while pin<pend
grain = DAFx_in(pin+1:pin+WLen).* w1;
%=====
f      = fft(fftshift(grain));
r      = abs(f);
phi   = angle(f);

delta_phi = omega + princarg(phi-phi0-omega);
phi0  = phi;
psi   = princarg(psi+delta_phi*tstretch_ratio);

ft    = (r.* exp(i*psi));
grain = fftshift(real(ifft(ft))).*w2;

%----- interpolation -----
grain2 = [grain;0];
grain3 = grain2(ix).*dx1+grain2(ix1).*dx;
% plot(grain);drawnow;
% =====
DAFx_out(pout+1:pout+lx) = DAFx_out(pout+1:pout+lx) + grain3;
pin   = pin + n1;
pout  = pout + n1;
end
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
toc

%----- listening and saving the output -----
%DAFx_in  = DAFx_in(WLen+1:WLen+L);
DAFx_out = DAFx_out(WLen+1:WLen+L) / max(abs(DAFx_out));
soundsc(DAFx_out, FS);
wavwrite(DAFx_out, FS, 'flute2_pitch_pv.wav');
```

This program is adapted from the time-stretching program using the FFT/IFFT approach. Here the grain is linearly interpolated before the reconstruction. The length of the interpolated grain is now $1x$ and will be overlapped and added with a hop size of $n1$ identical to the analysis hop size. In order to speed up the calculation of the interpolation, four vectors of length $1x$ are precalculated outside the main loop, which give the necessary parameters for the interpolation (ix , $ix1$, dx and $dx1$). As stated previously, the linear interpolation is not necessarily the best one, and will surely produce some foldover when the pitch shifting factor is greater than one. Other interpolation schemes can be inserted instead. Further pitch shifting techniques can be found in [QM98, Lar98, LD99b].

8.4.5 Stable/Transient Components Separation

This effect extracts “stable components” from a signal by selecting only points of the time-frequency representation that are considered as “stable in frequency” and eliminating all the other grains. Basic ideas can be found in [SL94]. From a musical point of view, one would think about getting only sine waves, and leave aside all the transient signals. However, this is not so: even with pure noise, the time-frequency analysis reveals some zones where we can have stable components. A pulse will also give an analysis where the instantaneous frequencies are the ones of the analyzing system and are very stable. Nevertheless this idea of separating a sound into two complementary sounds is indeed a musically good one. The result can be thought as an “etherization” of the sound for the stable one, and a “fractalization” for the transient one.

The algorithm for components separation is based on instantaneous frequency computation. The increment of the phase per sample for frequency bin k can be derived as

$$d\varphi(sR_a, k) = [\tilde{\varphi}(sR_a, k) - \tilde{\varphi}((s-1)R_a, k)] / R_a. \quad (8.45)$$

We will now sort out those points of a given FFT that give

$$d\varphi(sR_a, k) - d\varphi((s-1)R_a, k) < df, \quad (8.46)$$

where df is a preset value. From (8.45) and (8.46) we can derive the condition

$$\tilde{\varphi}(sR_a, k) - 2\tilde{\varphi}((s-1)R_a, k) + \tilde{\varphi}((s-2)R_a, k) < dfR_a. \quad (8.47)$$

From a geometrical point of view we can say that the value $\tilde{\varphi}(sR_a, k)$ should be in an angle dfR_a around the expected target value $\tilde{\varphi}_t(sR_a, k)$, as shown in Fig. 8.27.

It is important to note that the instantaneous frequencies may be out of the range of frequencies of the bin itself. The reconstruction performed by the inverse FFT takes only bins that follow this condition. In other words, only gaborets that follow the “frequency stability over time” condition are kept during the reconstruction. The following M-file 8.12 follows this guideline.

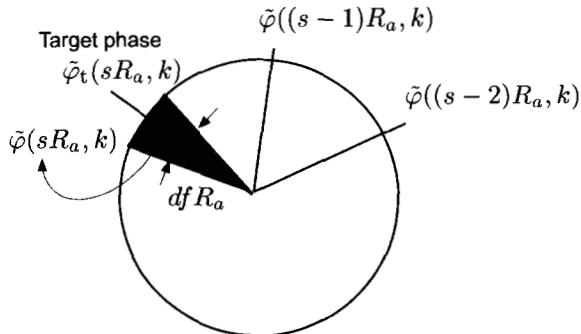


Figure 8.27 Evaluation of stable/unstable grains.

```

pout = 0;
pend = length(DAFx_in)-WLen;

while pin<pend
    grain = DAFx_in(pin+1:pin+WLen).* w1;
%=====
    f      = fft(fftshift(grain));
    theta = angle(f);
    dev   = princarg(theta-2*theta1+theta2);
% plot(dev);drawnow;
    ft    = f.*(abs(dev) < vtest);
    grain = fftshift(real(ifft(ft))).*w2;
    theta2 = theta1;
    theta1 = theta;
% =====
    DAFx_out(pout+1:pout+WLen) = ...
        DAFx_out(pout+1:pout+WLen) + grain;
    pin = pin + n1;
    pout = pout + n2;
end
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
toc

%---- listening and saving the output ----
%DAFx_in = DAFx_in(WLen+1:WLen+L);
DAFx_out = DAFx_out(WLen+1:WLen+L) / max(abs(DAFx_out));
soundsc(DAFx_out, FS);
wavwrite(DAFx_out, FS, 'redwheel_stable.wav');

```

So the algorithm for extraction of stable components performs the following steps:

1. Calculate the instantaneous frequency by making the derivative of the phase along the time axis.
2. Check if this frequency is within its “stable range”.
3. Use the frequency bin or not for the reconstruction.

The value of `vtest` is particularly important because it determines the level of the selection between stable and unstable bins.

The algorithm for transient components extraction is the same, except that we keep only bins where the condition (8.47) is not satisfied. So only two lines have to be changed according to

```

test =2           % new value for test
...
ft   = f*(abs(dev)>vtest); % new condition

```

In order to enhance the unstable grains the value `vtest` is usually higher for the transient extraction.

8.4.6 Mutation between Two Sounds

The idea is to calculate an arbitrary time-frequency representation from two original sounds and to reconstruct a sound from it. Some of these spectral mutations (see Fig. 8.28) give a flavor of cross-synthesis and morphing, a subject that will be discussed later, but are different from it, because here the effect is only incidental while in cross-synthesis hybridization of sounds is the primary objective. Further ideas can be found in [PE96]. There are different ways to calculate a new combined magnitude and phase diagram from the values of the original ones. As stated in section 8.3, an arbitrary image is not valid in the sense that it is not the time-frequency representation of a sound, which means that the result will be musically biased by the resynthesis scheme that we must use. Usually phases and magnitudes are calculated in an independent way, so that many combinations are possible. Not all of them are musically relevant, and the result also depends upon the nature of the sounds that are combined.

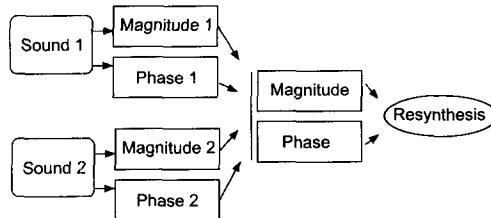


Figure 8.28 Basic principle of spectral mutations.

The following M-file 8.13 performs mutation between two sounds where the magnitude is coming from one sound and the phase from the other. Then only a few lines need to be changed to give different variations.

```

M-file 8.13 (VX_mutation.m)
% VX_mutation.m
%===== this program performs a mutation between two sounds,
%===== taking the phase of the first one and the modulus
%===== of the second one, and using:
%===== w1 and w2 windows (analysis and synthesis)
%===== WLen is the length of the windows
%===== n1 and n2: steps (in samples) for the analysis and synthesis

clear; clf

%---- user data -----
n1          = 512;
  
```

```

n2          = n1;
WLen       = 2048;
w1          = hanningz(WLen);
w2          = w1;
[DAFx_in1,FS] = wavread('x1.wav');
DAFx_in2    = wavread('x2.wav');

%----- initializations -----
L           = min(length(DAFx_in1),length(DAFx_in2));
DAFx_in1    = [zeros(WLen, 1); DAFx_in1; ...
    zeros(WLen-mod(L,n1),1)] / max(abs(DAFx_in1));
DAFx_in2    = [zeros(WLen, 1); DAFx_in2; ...
    zeros(WLen-mod(L,n1),1)] / max(abs(DAFx_in2));
DAFx_out    = zeros(length(DAFx_in1),1);

tic
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
pin = 0;
pout = 0;
pend = length(DAFx_in1) - WLen;

while pin<pend
    grain1 = DAFx_in1(pin+1:pin+WLen).* w1;
    grain2 = DAFx_in2(pin+1:pin+WLen).* w1;
%=====
    f1      = fft(fftshift(grain1));
    r1      = abs(f1);
    theta1 = angle(f1);
    f2      = fft(fftshift(grain2));
    r2      = abs(f2);
    theta2 = angle(f2);
    %---- the next two lines can be changed according to the effect
    r      = r1;
    theta = theta2;
    ft    = (r.* exp(i*theta));
    grain = fftshift(real(ifft(ft))).*w2;
% =====
    DAFx_out(pout+1:pout+WLen) = ...
        DAFx_out(pout+1:pout+WLen) + grain;
    pin   = pin + n1;
    pout  = pout + n2;
end
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
toc

%----- listening and saving the output -----

```

```
%DAFx_in = DAFx_in(WLen+1:WLen+L);
DAFx_out = DAFx_out(WLen+1:WLen+L) / max(abs(DAFx_out));
soundsc(DAFx_out, FS);
wavwrite(DAFx_out, FS, 'r1p2.wav');
```

Possible operations on the magnitude are:

1. Multiplication of the magnitudes $r=r1.*r2$ (so it is an addition in the dB scale). This corresponds to a logical “AND” operation, because one keeps all zones where energy is located.
2. Addition of the magnitude: the equivalent of a logical “OR” operation. However, this is different from mixing, because one only operates on the magnitude according to $r=r1+r2$.
3. Masking of one sound by the other is performed by keeping the magnitude of one sound if the other magnitude is under a fixed or relative threshold.

Operations on phase are really important for combinations of two sounds. Phase information is very important to ensure the validity (or quasivalidity) of time-frequency representations, and has an influence on the quality:

1. One can keep the phase from only one sound while changing the magnitude. This is a strong cue for the pitch of the resulting sound ($\theta=\theta_2$).
2. One can add the two phases. In this case we strongly alter the validity of the image (the phase turns with a mean double speed). We can also double the resynthesis hop size $n2=2*n1$.
3. One can take an arbitrary combination of the two phases but one should remember that phases are given modulo 2π (except if they have been unwrapped).
4. Design of an arbitrary variation of the phases.

As a matter of fact, these mutations are very experimental, and are very near to the construction of a true arbitrary time-frequency representation, but with some cues coming from the analysis of different sounds.

8.4.7 Robotization

This technique puts zero phase values on every FFT before reconstruction. The effect applies a fixed pitch onto a sound. Moreover, as it forces the sound to be periodic, many erratic and random variations are converted into robotic sounds. The sliding FFT of pulses where the analysis is taken at the time of these pulses will give a zero phase value for the phase of the FFT. This is a clear indication that putting a zero phase before an IFFT resynthesis will give a fixed pitch sound. This is reminiscent of the PSOLA technique, but here we do not make any assumption on the frequency of the analyzed sound and no marker has to be found. So zeroing the phase can be viewed from two points of view:

1. The result of an IFFT is a pulse-like sound and summing such grains at regular intervals gives a fixed pitch.
2. This can also be viewed as an effect of the reproducing kernel on the time-frequency representation: due to fact that the time-frequency representation now shows a succession of vertical lines with zero values in between, this will lead to a comb filter effect during resynthesis.

The following M-file 8.14 demonstrates the *robotization effect*.

M-file 8.14 (VX_robot.m)

```
% VX_robot.m
%===== this program performs a robotization of a sound, using:
%===== w1 and w2 windows (analysis and synthesis)
%===== WLen is the length of the windows
%===== n1 and n2: steps (in samples) for the analysis and synthesis

clear; clf

%---- user data ----
n1          = 441;
n2          = n1;
WLen        = 1024;
w1          = hanningz(WLen);
w2          = w1;
[DAFx_in, FS] = wavread('redwheel.wav');
L           = length(DAFx_in);
DAFx_in     = [zeros(WLen, 1); DAFx_in; ...
              zeros(WLen-mod(L,n1),1)] / max(abs(DAFx_in));

%---- initializations ----
DAFx_out    = zeros(length(DAFx_in),1);

tic
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
pin = 0;
pout = 0;
pend = length(DAFx_in)-WLen;
while pin<pend
    grain = DAFx_in(pin+1:pin+WLen).* w1;
%=====
    f      = fft(grain);
    r      = abs(f);
    grain = fftshift(real(ifft(r))).*w2;
% =====
    DAFx_out(pout+1:pout+WLen) = ...
        DAFx_out(pout+1:pout+WLen) + grain;
end
```

```

pin = pin + n1;
pout = pout + n2;
end
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
toc

%----- listening and saving the output -----
%DAFx_in = DAFx_in(WLen+1:WLen+L);
DAFx_out = DAFx_out(WLen+1:WLen+L) / max(abs(DAFx_out));
soundsc(DAFx_out, FS);
wavwrite(DAFx_out, FS, 'redwheel_robot.wav');

```

This is one of the shortest programs we can have, however, its effect is very strong. The only drawback is that the n_1 value in this program has to be an integer. The frequency of the robot is F_s/n_1 , where F_s is the sampling frequency. If the hop size is not an integer value, it is possible to use an interpolation scheme in order to dispatch the grain of two samples. This may happen if the hop size is calculated directly from a fundamental frequency value. An example is shown in Fig. 8.29.

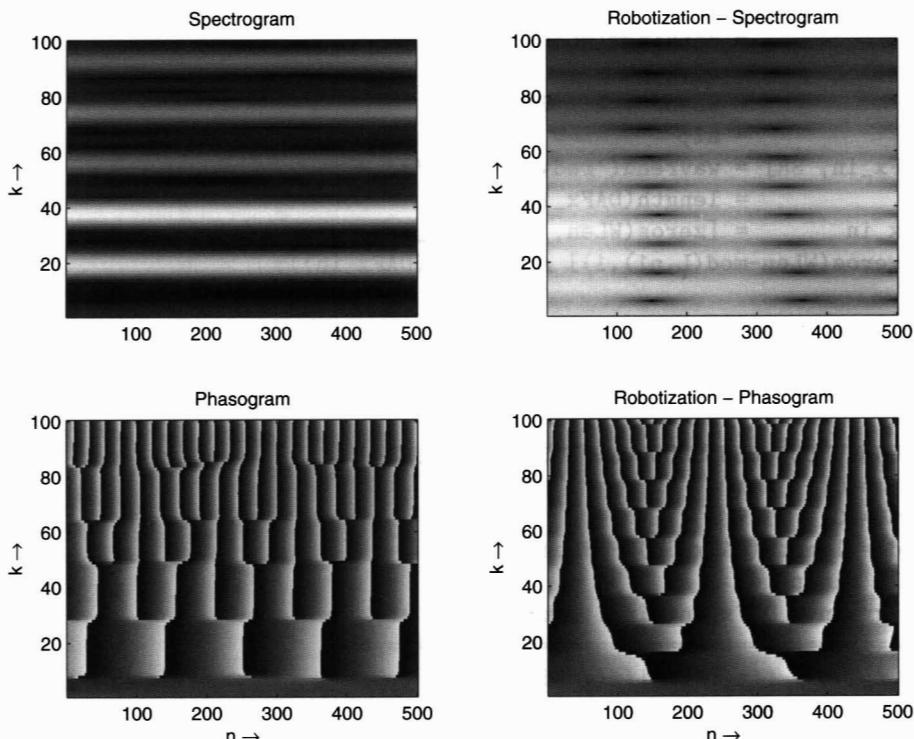


Figure 8.29 Example of robotization with a flute signal.

8.4.8 Whisperization

If we deliberately impose a random phase on a time-frequency representation, we can have a different behavior depending on the length of the window: if the window is quite large (for example, 2048 for a sampling rate of 44100 Hz), the magnitude will represent the behavior of the partials quite well and changes in phase will produce an uncertainty over the frequency. But if the window is small (e.g. 64 points), the spectral envelope will be enhanced and this will lead to a whispering effect. The M-file 8.15 implements the *whisperization* effect.

```
M-file 8.15 (VX_whisper.m)
% VX_whisper.m
%===== this program makes the whisperization of a sound,
%===== by randomizing the phase, using:
%===== w1 and w2 windows (analysis and synthesis)
%===== WLen is the length of the windows
%===== n1 and n2: steps (in samples) for the analysis and synthesis
clear; clf
%---- user data ----
WLen      = 512;
w1        = hannningz(WLen);
w2        = w1;
n1        = WLen/8; %   64;
n2        = n1;
[DAFx_in, SR] = wavread('redwheel.wav');
L          = length(DAFx_in);
DAFx_in    = [zeros(WLen, 1); DAFx_in; ...
             zeros(WLen-mod(L,n1),1)] / max(abs(DAFx_in));

%---- initializations ----
DAFx_out   = zeros(length(DAFx_in),1);

tic
%%%%%%%%%%%%%
pin = 0;
pout = 0;
pend = length(DAFx_in) - WLen;
while pin<pend
    grain = DAFx_in(pin+1:pin+WLen).* w1;
%=====
    f      = fft(fftshift(grain));
    r      = abs(f);
    phi   = 2*pi*rand(WLen,1);
    ft    = (r.* exp(i*phi));
    grain = fftshift(real(ifft(ft))).*w2;
% =====
```

```

DAFx_out(pout+1:pout+WLen) = ...
    DAFx_out(pout+1:pout+WLen) + grain;
pin   = pin + n1;
pout  = pout + n2;
end
%%%%%%%%%%%%%
toc

%----- listening and saving the output -----
%DAFx_in = DAFx_in(WLen+1:WLen+L);
DAFx_out = DAFx_out(WLen+1:WLen+L) / max(abs(DAFx_out));
soundsc(DAFx_out, SR);
wavwrite(DAFx_out, SR, 'whisper2.wav');

```

It is also possible to make a random variation of the magnitude and keep the phase. An example is shown in Fig. 8.30. This gives another way to implement whisperization, which can be achieved by the following MATLAB kernel:

```

%=====
f      = fft(fftshift(grain));
r      = abs(f).*randn(lfen,1);
phi   = angle(f);
ft     = (r.* exp(i*phi));
grain = fftshift(real(ifft(ft))).*w2;
% =====

```

8.4.9 Denoising

A musician may want to emphasize some specific areas of a spectrum and lower the noise within a sound. Though this is achieved more perfectly by the use of a sinusoidal model (see Chapter 10) but another approach is the use of denoising algorithms. The algorithm we describe uses a nonlinear spectral subtraction technique [Vas96]. Further techniques can be found in [Cap94]. A time-frequency analysis and resynthesis are performed, with an extraction of the magnitude and phase information. The phase is kept as it is, while the magnitude is processed in such a way that it keeps the high-level values while attenuating the lower ones, in such a way as to attenuate the noise. This can also be seen as a bank of noise gates on different channels, because on each bin we perform a nonlinear operation. The denoised magnitude vector $X_d(n, k) = f(X(n, k))$ of the denoised signal is then the output of a noise gate with a nonlinear function $f(x)$. A basic example of such a function is $f(x) = x^2/(x + c)$, which is shown in Fig. 8.31. It can also be seen as the multiplication of the magnitude vector by a correction factor $x/(x + c)$. The result of such a waveshaping function on the magnitude spectrum keeps the high values of the magnitude and lowers the small ones. Then the phase of the initial signal is reintroduced and the sound is reconstructed by overlapping grains with the help of an IFFT. The following M-file 8.16 follows this guideline.

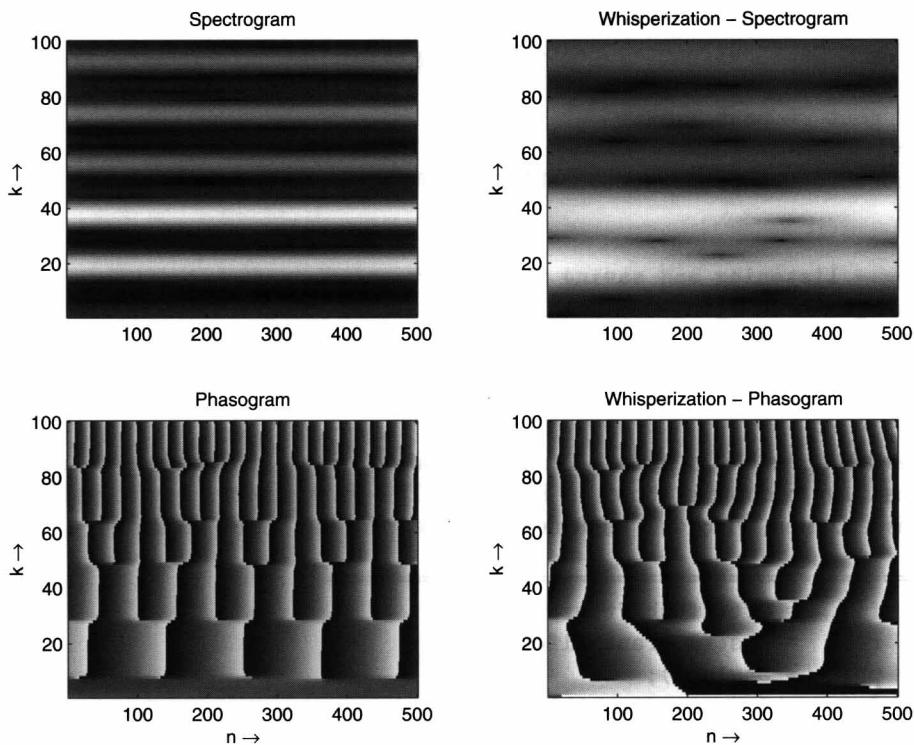


Figure 8.30 Example of whisperization with a flute signal.

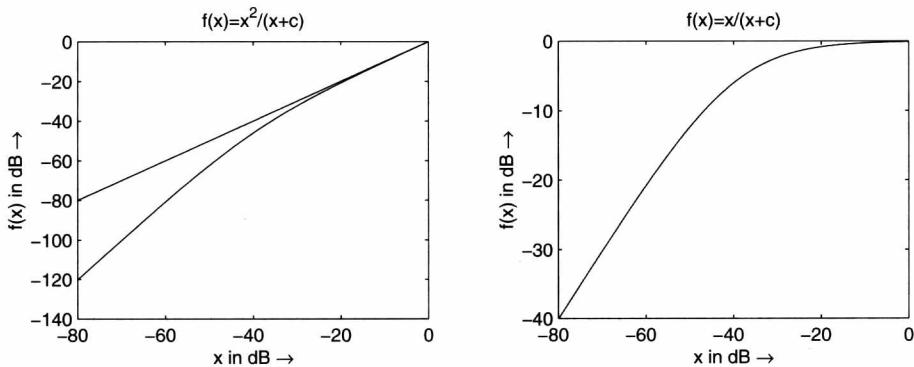


Figure 8.31 Nonlinear function for a noise gate.

M-file 8.16 (VX_denoise.m)

```
% VX_denoise.m
```

```
===== this program makes a denoising of a sound, using:  
===== w1 and w2 windows (analysis and synthesis)
```

```
%==== WLlen is the length of the windows
%==== n1 and n2: steps (in samples) for the analysis and synthesis
clear; clf
%---- user data ----
n1          = 512;
n2          = n1;
WLen        = 2048;
w1          = hanningz(WLen);
w2          = w1;
[DAFx_in, FS] = wavread('x1.wav');
%---- initializations ----
L = length(DAFx_in);
DAFx_in = [zeros(WLen, 1); DAFx_in; ...
    zeros(WLen-mod(L,n1),1)] / max(abs(DAFx_in));
WLen2       = WLen/2;
coef        = 0.01;
freq        = (0:1:299)/WLen*44100;
DAFx_out    = zeros(length(DAFx_in),1);

tic
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
pin = 0;
pout = 0;
pend = length(DAFx_in) - WLen;

while pin<pend
    grain = DAFx_in(pin+1:pin+WLen).* w1;
%=====
    f      = fft(grain);
    r      = abs(f)/WLen2;
    ft    = f.*r./(r+coef);
    grain = (real(ifft(ft))).*w2;
%=====
    DAFx_out(pout+1:pout+WLen) = ...
        DAFx_out(pout+1:pout+WLen) + grain;
    pin = pin + n1;
    pout = pout + n2;
end
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
toc
%---- listening and saving the output ----
%DAFx_in = DAFx_in(WLen+1:WLen+L);
DAFx_out = DAFx_out(WLen+1:WLen+L);
soundsc(DAFx_out, FS);
wavwrite(DAFx_out, FS, 'x1_denoise.wav');
```

An example is shown in Fig. 8.32. It is of course possible to introduce different noise gate functions instead of the simple ones we have chosen.

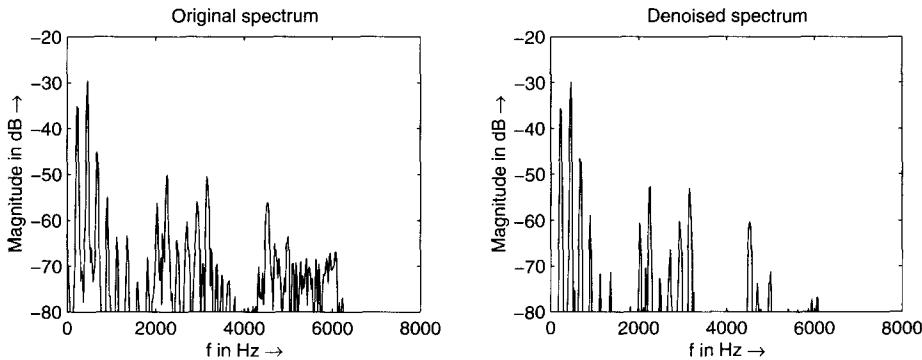


Figure 8.32 The left plot shows the windowed FFT of a flute sound. The right plot shows the same FFT after noise gating each bin using the $r/(r+\text{coef})$ gating function with $c = 0.01$.

Denoising in itself has many variations depending on the application:

1. Denoising from a tape recorder usually starts from the analysis of a noisy sound coming from a recording of silence. This gives a gaboret for the noise shape, so that the nonlinear function will be different for each bin, and can be zero under this threshold.
2. The noise level can be estimated in a varying manner. For example, one can estimate a noise threshold which can be spectrum dependent. This usually involves spectral estimation techniques (with the help of LPC or cepstrum), which will be seen later.
3. One can also try to evaluate a level of noise on successive time instances in order to decrease pumping effects.
4. In any case, these algorithms involve nonlinear operations and as such can produce artifacts. One of them is the existence of small grains that remain outside the silence unlike the previous noise (spurious components). The other artifact is that noise can sometimes be a useful component of a sound and will be suppressed as undesirable noise.

8.5 Conclusion

The starting point of this chapter was the computation of a time-frequency representation of a sound, to manipulate this representation and reproduce a sound. At first sight this may appear as an easy task, but we have seen that the basis

for this time-frequency processing needs a careful description of the fundamentals, because the term vocoder can cover different implementations. We also explained that the arbitrary manipulation of time-frequency representations renders images in a way that they are no longer time-frequency representations of “real” sounds. This phenomenon leads to artifacts, which cannot be avoided.

Digital audio effects described in this chapter only perform manipulations of these time-frequency representations. These effects exclude the extraction of resonances, which will be the subject of the next chapter, and high-level processing such as the extraction of sinusoids and noise. For example, the mentioned bank of filters does not assume any parametric model of the sound. Nevertheless such effects are numerous and diverse. Some of them have brought new solutions to well-known techniques such as filtering. Pitch shifting and time stretching have shown their central place in the phase vocoder approach, which is another implementation possibility independent of the time processing approach shown in Chapter 7. Their was a clear need for a clarification of the phase vocoder approach in this domain. Though it has been known for years, we have provided a general framework and simple implementations upon which more complex effects may be built. Some of them can reduce the phasiness of the process or perform special high level processing on transients. Other digital audio effects have been described that fit well under the name “mutations”. They are based on modifying the magnitude and phase of one or two time-frequency representations. They put a special flavor on sounds, which musicians characterize as granulation, robotization, homogenization, purification, metallization and so on. Once again, the goal of this chapter is to give a general framework and unveil some of the basic implementations of these alterations of sound, which can be extended to more complex modifications at will.

This chapter is a good starting point for the computer–human interface and the digital control of effects, but this is beyond the scope of this chapter. Nevertheless it must be said that this part is crucial in the design of a digital audio effect. We refer here to Chapter 12 to see the prospective view it requires.

As final remark, one can say that no digital audio effect and time-frequency processing in particular would exist without a sound. Only a good adaptation of the sound with the effect can give rise to musical creativity. This is the reason why some of the basic algorithms presented put in the hands of creative musicians and artists can give better results than much more complex algorithms in the hands of conventional persons.

Bibliography

- [AD93] D. Arfib and N. Delprat. Musical transformations using the modification of time-frequency images. *Computer Music Journal*, 17(2):66–72, 1993.
- [Cap94] O. Cappé. Elimination of the musical noise phenomenon with the ephraim and malah noise suppressor. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 2:345–349, 1994.

- [CGT89] J.M. Combes, A. Grossmann, and Ph. Tchamitchan (eds). *Wavelets. Time Frequency Methods and Phase Space*. Springer-Verlag, 2nd edition, 1989.
- [Chu92] C.H. Chui. *An Introduction to Wavelets*. Academic Press, 1992.
- [CR83] R.E. Crochiere and L.R. Rabiner. *Multirate Digital Signal Processing*. Prentice-Hall, 1983.
- [Cro80] R.E. Crochiere. A weighted overlap-add method of short-time fourier analysis/synthesis. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 281(1):99–102, 1980.
- [Fer99] A.J.S. Ferreira. An odd-DFT based approach to time-scale expansion of audio signals. *IEEE Trans. on Speech and Audio Processing*, 7(4):441–453, 1999.
- [GBA00] A. De Götzen, N. Bernadini, and D. Arfib. Traditional (?) implementations of a phase vocoder: The tricks of the trade. In *Proc. DAFX-00 Conference on Digital Audio Effects*, pp. 37–43, Verona, December 2000.
- [Lar98] J. Laroche. Time and pitch scale modifications of audio signals. In M. Kahrs and K.-H. Brandenburg (eds), *Applications of Digital Signal Processing to Audio and Acoustics*, pp. 279–309. Kluwer, 1998.
- [LD99a] J. Laroche and M. Dolson. Improved phase vocoder time-scale modification of audio. *IEEE Trans. on Speech and Audio Processing*, 7(3):323–332, 1999.
- [LD99b] J. Laroche and M. Dolson. New phase-vocoder techniques for real-time pitch shifting, chorusing, harmonizing, and other exotic audio modifications. *J. Audio Eng. Soc.*, 47(11):928–936, 1999.
- [PE96] L. Polansky and T. Erbe. Spectral mutation in soundhack. *Computer Music Journal*, 20(1):92–101, Spring 1996.
- [Por76] M.R. Portnoff. Implementation of the digital phase vocoder using the fast fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 24(3):243–248, June 1976.
- [QC93] S. Quiian and D. Chen. Discrete gabor transform. *IEEE Transactions on Signal Processing*, 41(7):2429–2438, 1993.
- [QM98] T.F. Quatieri and R.J. McAulay. Audio signal processing based on sinusoidal analysis/synthesis. In M. Kahrs and K.-H. Brandenburg (eds), *Applications of Digital Signal Processing to Audio and Acoustics*, pp. 343–416. Kluwer, 1998.
- [SL94] Z. Settel and C. Lippe. Real-time musical applications using the FFT-based resynthesis. In *Proc. International Computer Music Conference*, 1994.

- [Vas96] S.V. Vaseghi. *Advanced Signal Processing and Digital Noise Reduction*. Wiley & Teubner, 1996.
- [WR90] J. Wexler and S. Raz. Discrete gabor expansions. *Signal Processing*, 21(3):207–220, 1990.
- [Zöl97] U. Zölzer. *Digital Audio Signal Processing*. John Wiley & Sons, Ltd, 1997.

Chapter 9

Source-Filter Processing

D. Arfib, F. Keiler, U. Zölzer

9.1 Introduction

Time-frequency representations give the evolution over time of a spectrum calculated from temporal frames. The notion of the spectral envelope extracted from such representations mostly comes from the voice production and recognition system: the voice production uses vocal chords as an excitation and the mouth and nose as resonator system or anti-resonator. Voiced signals (vowels) produce a harmonic spectrum on which a spectral envelope is superimposed. This fact about voice strongly influences our way of recognizing other sounds, whether because of the ear or the brain: we are looking for such a spectral envelope as a cue to the identification or classification of sounds. This excitation-resonance model is also called source-filter model in the literature. Thus we can understand why the vocoding effect, which is the cross-synthesis of a musical instrument with voice, is so attractive for the ear and so resistant to approximations. We will make use of a source-filter model for an audio signal and modify this model in order to achieve different digital audio effects.

However, the signal processing problem of extracting a spectral envelope from a spectrum is generally badly conditioned. If the sound is purely harmonic we could say that the spectral envelope is the curve that passes through the points related to these harmonics. This leaves two open questions: how to retrieve these exact values of these harmonics, and what kind of interpolation scheme should we use for the completion of the curve in between these points? But, more generally, if the sound contains inharmonic partials or a noisy part, this definition no longer holds and the notion of a spectral envelope is then completely dependent on the definition of what belongs to the excitation and what belongs to the resonance. In a way it is more a “envelope recognition” problem than a “signal processing” one.

With this in mind we will state that a spectral envelope is a smoothing of a

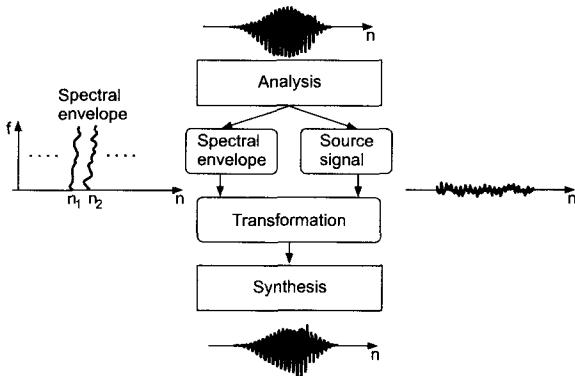


Figure 9.1 Spectral processing based on time-varying spectral envelopes and source signals. The analysis performs a source and filter separation.

spectrum, which tends to leave aside the spectral lines structure while preserving the general form of the spectrum. Three techniques with many variants can be used for the estimation of the spectral envelope:

1. The **channel vocoder** uses frequency bands and performs estimations of the amplitude of the signal inside these bands and thus the spectral envelope.
2. **Linear prediction** estimates an all-pole filter that matches the spectral content of a sound. When the order of this filter is low, only the formants are taken, hence the spectral envelope.
3. **Cepstrum** techniques perform smoothing of the logarithm of the FFT spectrum (in decibels) in order to separate this curve into its slow varying part (the spectral envelope) and its quickly varying part (the source signal).

For each of these techniques, we will describe the fundamental algorithms in section 9.2 which allow the calculation of the spectral envelope and the source signal in a frame oriented approach, as shown in Fig. 9.1. Then transformations are applied to the spectral envelope and/or the source signal and a synthesis procedure reconstructs the output sound. Some basic transformations are introduced in section 9.3. The separation of a source and a filter is only one of the features we can extract from a sound, or more precisely from a time-frequency representation. The final section 9.4 describes the extraction of other very important features such as the pitch, the centroid, and the harmonic/noise balance, which can be used to modify control parameters for digital audio effects.

9.2 Source-Filter Separation

Digital audio effects based on source-filter processing extract the spectral envelope and the source (excitation) signal from an input signal, as shown in Fig. 9.2. The

input signal is whitened by the filter $1/H_1(z)$, which is derived from the spectral envelope of the input signal. In signal processing terms, the spectral envelope is given by the magnitude response $|H_1(f)|$ or its logarithm $\log |H_1(f)|$ in dB. This leads to extraction of the source signal $e_1(n)$ which can be further processed, for example, by time stretching or pitch shifting algorithms. The processed source signal is then finally filtered by $H_2(z)$. This filter is derived from the modified spectral envelope of the input signal or another source signal.

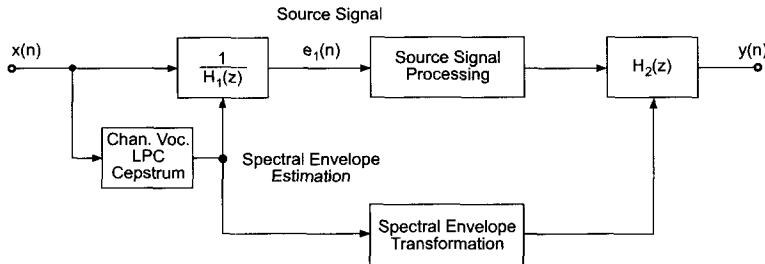


Figure 9.2 Spectrum estimation (Channel vocoder, Linear Predictive Coding or Cepstrum) and source signal extraction for individual processing.

9.2.1 Channel Vocoder

If we filter a sound with a bank of bandpass filters and calculate the RMS value for each bandpass signal, we can obtain an estimation of the spectral envelope (see Fig. 9.3). The parameters of the filters for each channel will of course affect the precision of the measurement, as well as the delay between the sound input and the spectral calculation. The RMS calculation parameters are also a compromise between a good definition and an acceptable delay and trail effect. The spectral estimation is valid around the center frequency of the filters. Thus the more channels there are, the more frequency points of the spectral envelope are estimated. The filter bank can be defined on a linear scale, in which case every filter of the filter bank can be equivalent in terms of bandwidth. It can also be defined on a logarithmic scale. In this case, this approach is more like an “equalizer system” and the filters, if given in the time domain, are scaled versions of a mother filter.

The channel vocoder algorithm shown in Fig. 9.3 works in the time domain. There is, however, a possible derivation where it is possible to calculate the spectral envelope from the FFT spectrum, thus directly from the time-frequency representations. A channel can be represented in the frequency domain, and the energy of an effective channel filter can be seen as the sum of the elementary energies of each bin weighted by this channel filter envelope. The amplitude coming out of this filter is then the square root of these energies.

In the case of filters with equally-spaced channel stacking (see Fig. 9.3b), it is even possible to use a short-cut for the calculation of this spectral envelope: the spectral envelope is the square root of the filtered version of the squared amplitudes. This

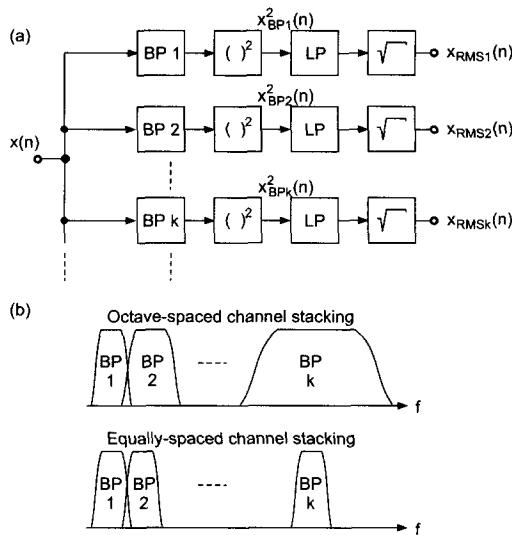


Figure 9.3 (a) Channel vocoder and (b) frequency stacking.

computation can be performed by a circular convolution $Y(k) = \sqrt{|X(k)|^2 * w(k)}$ in the frequency domain, where $w(k)$ may be a Hanning window function. The circular convolution is accomplished by another FFT/IFFT filtering algorithm. The result is a spectral envelope, which is a smoothing of the FFT values. An example is shown in Fig. 9.4.

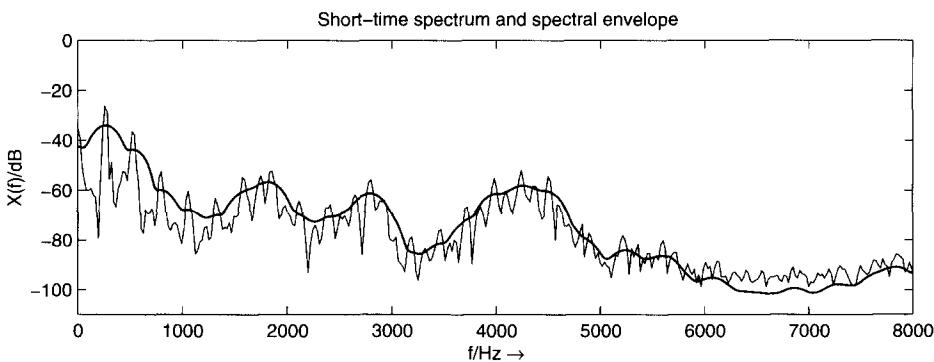


Figure 9.4 Spectral envelope computation with a channel vocoder.

The following M-file 9.1 defines channels in the frequency domain and calculates the energy in dB inside successive channels of that envelope.

M-file 9.1 (specenvcv.m)

```
WLen=2048; w=hanningz(WLen);
```

```

buf=y(offset:offset+WLen-1).*w;
f=fft(buf)/(WLen/2);
freq=(0:1:WLen-1)/WLen*44100;
flog=20*log10(0.00001+abs(f));
% Frequency window
nob=input('number of bins must be even = ');
w1=hanningz(nob);w1=w1./sum(w1);
f_channel=[zeros((WLen-nob)/2,1);w1;zeros((WLen-nob)/2,1)];
% FFT of frequency window
fft_channel=fft(fftshift(f_channel));
f2=f.*conj(f); % Squared FFT values
% Circ. Convolution by FFT-Multiplication-IFFT
energy=real(ifft(fft(f2).*fft_channel));
flog_rms=10*log10(abs(energy));
%10 indicates a combination with sqrt operation
subplot(2,1,1);plot(freq,flog,freq,flog_rms);
ylabel('X(f)/dB');
xlabel('f/Hz \rightarrow');axis([0 8000 -110 0]);
title('Short-time spectrum and spectral envelope');

```

The program starts with the calculation of the FFT of a windowed frame, where *w* is a Hanning window in this case. The vector *y* contains the sound and a buffer *buf* contains a windowed segment. In the second part of this program *fchannel* represents the envelope of the channel with a FFT representation. Here it is a Hanning window of width *nob*, which is the number of frequency bins. The calculation of the weighted sum of the energies inside a channel is performed by a convolution calculation of the energy pattern and the channel envelope. Here, we use a circular convolution with an FFT-IFFT algorithm to easily retrieve the result for all channels. In a way it can be seen as a smoothing of the energy pattern. The only parameter is the envelope of the channel filter, hence the value of *nob* in this program. The fact that it is given in bins and that it should be even is only for the simplification of the code. The bandwidth is given by $nob \cdot \frac{f_s}{N}$ (*N* is the length of the FFT).

9.2.2 Linear Predictive Coding (LPC)

One way to estimate the spectral envelope of a sound is directly based on a simple sound production model. In this model, the sound is produced by passing an excitation source (source signal) through a synthesis filter, as shown in Fig. 9.5. The filter models the resonances and has therefore only poles. Thus, this all-pole filter represents the spectral envelope of the sound. This model works well for speech, where the synthesis filter models the human vocal tract, while the excitation source consists of pulses plus noise [Mak75]. For voiced sounds the periodicity of the pulses determines the pitch of the sound while for unvoiced sounds the excitation is noise-like.

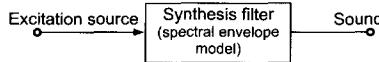


Figure 9.5 Sound production model: the synthesis filter represents the spectral envelope.

The retrieval of the spectral envelope from a given sound at a given time is based on the estimation of the all-pole synthesis filter mentioned previously. This approach is widely used for speech coding and is called linear predictive coding (LPC) [Mak75, MG76].

Analysis/Synthesis Structure

In LPC the current input sample $x(n)$ is approximated by a linear combination of past samples of the input signal. The prediction of $x(n)$ is computed using an FIR filter by

$$\hat{x}(n) = \sum_{k=1}^p a_k x(n-k) \quad (9.1)$$

where p is the *prediction order* and a_k are the prediction coefficients. The difference between the original input signal $x(n)$ and its prediction $\hat{x}(n)$ is evaluated by

$$e(n) = x(n) - \hat{x}(n) = x(n) - \sum_{k=1}^p a_k x(n-k). \quad (9.2)$$

The difference signal $e(n)$ is called *residual* or *prediction error* and its calculation is depicted in Fig. 9.6 where the transversal (direct) FIR filter structure is used.

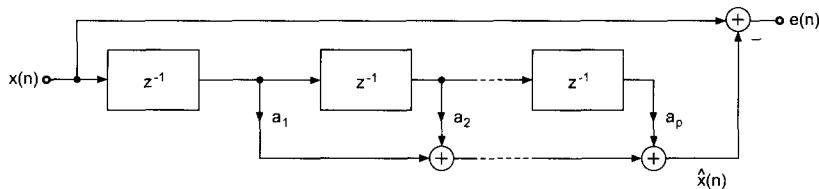


Figure 9.6 Transversal FIR filter structure for the prediction error calculation.

With the z -transform of the *prediction filter*

$$P(z) = \sum_{k=1}^p a_k z^{-k}, \quad (9.3)$$

Equation (9.2) can be written in the z -domain as

$$E(z) = X(z) - \hat{X}(z) = X(z)[1 - P(z)]. \quad (9.4)$$

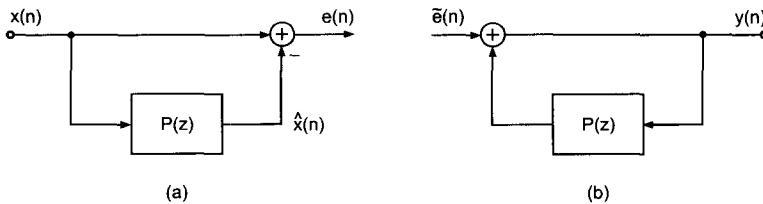


Figure 9.7 LPC structure with feed forward prediction. (a) Analysis, (b) Synthesis.

Figure 9.7(a) illustrates the last equation. The illustrated structure is called *feed forward prediction* where the prediction is calculated in the forward direction from the input signal.

Defining the *prediction error filter* or *inverse filter*

$$A(z) = 1 - P(z) = 1 - \sum_{k=1}^p a_k z^{-k}, \quad (9.5)$$

the prediction error is obtained as

$$E(z) = X(z)A(z). \quad (9.6)$$

The sound signal is recovered by using the *excitation signal* $\tilde{e}(n)$ as input to the all-pole filter

$$H(z) = \frac{1}{A(z)} = \frac{1}{1 - P(z)}. \quad (9.7)$$

This yields the output signal

$$Y(z) = \tilde{E}(z) \cdot H(z) \quad (9.8)$$

where $H(z)$ can be realized with the FIR filter $P(z)$ in a feedback loop as shown in Fig. 9.7(b). If the residual $e(n)$, which is calculated in the analysis stage, is fed directly into the synthesis filter, the input signal $x(n)$ will be ideally recovered.

The IIR filter $H(z)$ is termed *synthesis filter* or *LPC filter* and represents the spectral model – except for a gain factor – of the input signal $x(n)$. As mentioned previously, this filter models the time-varying vocal tract in the case of speech signals.

With optimal filter coefficients, the residual energy is minimized. This can be exploited for efficient coding of the input signal where the quantized residual $\tilde{e}(n) = Q\{e(n)\}$ is used as excitation to the LPC filter.

Figure 9.8 shows an example where for a short block of a speech signal an LPC filter of order $p = 50$ is computed. In the left plot the time signal is shown while the right plot shows both the spectra of the input signal and of the LPC filter $H(z)$. In this example the autocorrelation method is used to calculate the LPC coefficients. The MATLAB code for this example is given by M-file 9.2 (the used function `calc_lpc` will be explained later).

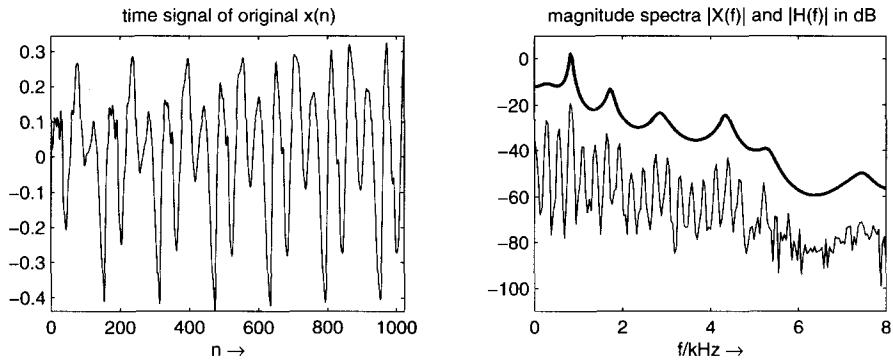


Figure 9.8 LPC example for the female utterance “la” with prediction order $p = 50$, original signal and LPC filter.

M-file 9.2 (figure9_08.m)

```

fname='la.wav';
n0=5000; %start index
N=1024; %block length
Nfft=1024; % FFT length
p=50; %prediction order
n1=n0+N-1; %end index

[xin,Fs]=wavread(fname,[n0 n1]);
x=xin(:,1)'; % row vector of left channel
win=hamming(N)'; % window for input block

a=calc_lpc(x.*win,p); % calculate LPC coeffs
% a=[1, -a_1, -a_2, ..., -a_p]

Omega=(0:Nfft-1)/Nfft*Fs/1000; % frequencies in kHz
offset=20*log10(2/Nfft); % offset of spectrum in dB
A=20*log10(abs(fft(a,Nfft)));
H=-A+offset;
X=20*log10(abs(fft(x.*win,Nfft)));
X=X+offset;

```

Calculation of the Filter Coefficients

To find an all-pole filter which models a considered sound well, different approaches may be taken. Some common methods compute the filter coefficients from a block of the input signal $x(n)$. These methods are namely the autocorrelation method [Mak75, Orf90], the covariance method [Mak75, MG76], and the Burg algorithm [Mak77, Orf90]. Since both the autocorrelation method and the Burg algorithm compute the lattice coefficients, they are guaranteed to produce stable synthesis

filters while the covariance method may yield unstable filters.

Now we briefly describe the autocorrelation method which minimizes the energy of the prediction error $e(n)$. With the prediction error $e(n)$ defined in (9.2), the prediction error energy is¹

$$E_p = E \{ e^2(n) \}. \quad (9.9)$$

Setting the partial derivatives of E_p with respect to the filter coefficients a_i ($i = 1, \dots, p$) to zero leads to

$$\frac{\partial E_p}{\partial a_i} = 2E \left\{ e(n) \cdot \frac{\partial e(n)}{\partial a_i} \right\} \quad (9.10)$$

$$= -2E \{ e(n)x(n-i) \} \quad (9.11)$$

$$= -2E \left\{ \left[x(n) - \sum_{k=1}^p a_k x(n-k) \right] x(n-i) \right\} = 0 \quad (9.12)$$

$$\Leftrightarrow \sum_{k=1}^p a_k E \{ x(n-k)x(n-i) \} = E \{ x(n)x(n-i) \}. \quad (9.13)$$

Equation (9.13) is a formulation of the so-called *normal equations* [Mak75]. The autocorrelation sequence for a block of length N is defined by

$$r_{xx}(i) = \sum_{n=i}^{N-1} u(n)u(n-i) \quad (9.14)$$

where $u(n) = x(n) \cdot w(n)$ is a windowed version of the considered block $x(n)$, $n = 0, \dots, N-1$. Normally a Hamming window is used [O'S00]. The expectation values in (9.13) can be replaced by their approximations using the autocorrelation sequence, which gives the normal equations²

$$\sum_{k=1}^p a_k r_{xx}(i-k) = r_{xx}(i) \quad , i = 1, \dots, p. \quad (9.15)$$

The filter coefficients a_k ($k = 1, \dots, p$) which model the spectral envelope of the used segment of $x(n)$ are obtained by solving the normal equations. An efficient solution of the normal equations is performed by the Levinson-Durbin recursion [Mak75].

As explained in [Mak75], minimizing the residual energy is equivalent to finding a best spectral fit in the frequency domain, if the gain factor is ignored. Thus the input signal $x(n)$ is modeled by the filter

$$H_g(z) = G \cdot H(z) = \frac{G}{1 - \sum_{k=1}^p a_k z^{-k}} \quad (9.16)$$

¹With the expectation value $E \{ \cdot \}$.

²The multiplication of the expectation values by the block length N does not have any effect on the normal equations.

where G denotes the gain factor. With this modified synthesis filter the original signal is modeled using a white noise excitation with unit variance. For the auto-correlation method the gain factor is defined by [Mak75]

$$G^2 = r_{xx}(0) - \sum_{k=1}^p a_k r_{xx}(k) \quad (9.17)$$

with the autocorrelation sequence given in (9.14). Hence the gain factor depends on the energy of the prediction error. If $|H_g(e^{j\Omega})|^2$ models the power spectrum $|X(e^{j\Omega})|^2$, the prediction error power spectrum is a flat spectrum with $|E(e^{j\Omega})|^2 = G^2$. The inverse filter $A(z)$ to calculate the prediction error is therefore also called the “whitening filter” [Mak75]. The MATLAB code of the function `calc_lpc` for the calculation of the prediction coefficients and the gain factor using the autocorrelation method is given by M-file 9.3.

M-file 9.3 (calc_lpc.m)

```
function [a,g]=calc_lpc(x,p)
% calculate LPC coeffs via autocorrelation method
% x: input signal, p: prediction order

R=xcorr(x,p);          % autocorrelation sequence R(k) with k=-p,...,p
R(1:p)=[];              % delete entries for k=-p,...,-1
if norm(R)~=0
    a=levinson(R,p);    % Levinson-Durbin recursion
% a=[1, -a_1, -a_2,..., -a_p]
else
    a=[1, zeros(1,p)];
end
R=R(:); a=a(:);      % row vectors
g=sqrt(sum(a.*R));   % gain factor
```

Notice that normally the MATLAB function `lpc` can be used, but with MATLAB release 12 (version 6) this function has been changed.

Figure 9.9 shows the prediction error and the estimated spectral envelope for the input signal shown in Figure 9.8. It can clearly be noticed that the prediction error has strong peaks occurring with the period of the fundamental frequency of the input signal. We can make use of this property of the prediction error signal for computing the fundamental frequency. The fundamental frequency and its pitch period can deliver pitch marks for PSOLA time stretching or pitch shifting algorithms or other applications. The corresponding MATLAB code is given by M-file 9.4.

M-file 9.4 (figure9_09.m)

```
fname='la.wav';
n0=5000;    %start index
N=1024;     %block length
Nfft=1024;   % FFT length
```

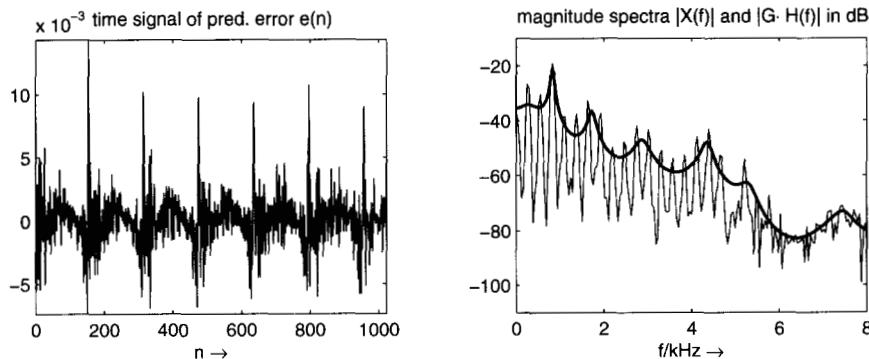


Figure 9.9 LPC example for the female utterance “la” with prediction order $p = 50$, prediction error and spectral envelope.

```

p=50;           %prediction order
n1=n0+N-1;    %end index
pre=p;          %filter order= no. of samples required before n0

[xin,Fs]=wavread(fname,[n0-pre n1]);
xin=xin(:,1)';
win=hamming(N)';
x=xin((1:N)+pre); % block without pre-samples

[a,g]=calc_lpc(x.*win,p); % calculate LPC coeffs and gain
% a=[1, -a_1, -a_2,..., -a_p]
g_db=20*log10(g)      % gain in dB

ein=filter(a,1,xin); % pred. error
e=ein((1:N)+pre);   % without pre-samples
Gp=10*log10(sum(x.^2)/sum(e.^2))  % prediction gain

Omega=(0:Nfft-1)/Nfft*Fs/1000; % frequencies in kHz
offset=20*log10(2/Nfft);        % offset of spectrum in dB
A=20*log10(abs(fft(a,Nfft)));
H_g=-A+offset+g_db;            % spectral envelope
X=20*log10(abs(fft(x.*win,Nfft)));
X=X+offset;

```

Thus for the computation of the prediction error over the complete block length additional samples of the input signal are required. The calculated prediction error signal $e(n)$ is equal to the source or excitation which has to be used as input to the synthesis filter $H(z)$ to recover the original signal $x(n)$. For this example the

prediction gain, defined as

$$G_p = \frac{\sum_{n=0}^{N-1} x^2(n)}{\sum_{n=0}^{N-1} e^2(n)}, \quad (9.18)$$

has the value 38 dB, and the gain factor is $G = -23$ dB.

Figure 9.10 shows spectra of LPC filters at different filter orders for the same signal block as in Fig. 9.8. The bottom line shows the spectrum of the signal segment where only frequencies below 8 kHz are depicted. The other spectra in this plot show the results using the autocorrelation method with different prediction orders. For clarity reasons these spectra are plotted with different offsets. It is obvious that for an increasing prediction order the spectral model gets better although the prediction gain only increases from 36.6 dB ($p = 10$) to 38.9 dB ($p = 120$).

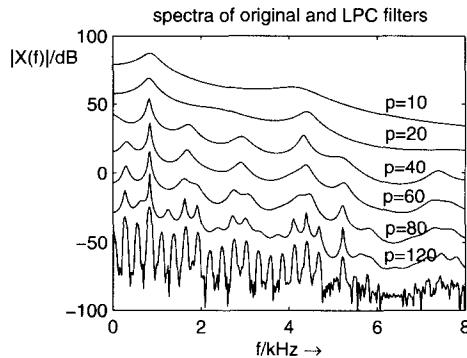


Figure 9.10 LPC filter spectra for different prediction orders for the female utterance “la”.

In summary, the LPC method delivers a source-filter model and allows the determination of pitch marks or the fundamental frequency of the input signal.

9.2.3 Cepstrum

The cepstrum (backward spelling of “spec”) method allows the estimation of a spectral envelope starting from the FFT values $X(k)$ of a windowed frame $x(n)$. Zero padding and Hanning, Hamming or Gaussian windows can be used depending on the number of points used for the spectral envelope estimation. An introduction to the basics of cepstrum based signal processing can be found in [OS75]. The cepstrum is calculated from the discrete Fourier transform

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{kn} = |X(k)|e^{j\varphi_x(k)}, \quad k = 0, 1, \dots, N-1 \quad (9.19)$$

by taking the logarithm

$$\hat{X}(k) = \log X(k) = \log |X(k)| + j\varphi_x(k) \quad (9.20)$$

and performing an IFFT of $\hat{X}(k)$, which yields the *complex cepstrum*

$$\hat{x}(n) = \frac{1}{N} \sum_{k=0}^{N-1} \hat{X}(k) W_N^{-kn}. \quad (9.21)$$

The *real cepstrum* is derived from the real part of (9.20) given by

$$\hat{X}_R(k) = \log |X(k)| \quad (9.22)$$

and performing an IFFT of $\hat{X}_R(k)$, which leads to the *real cepstrum*

$$c(n) = \frac{1}{N} \sum_{k=0}^{N-1} \hat{X}_R(k) W_N^{-kn}. \quad (9.23)$$

Since $\hat{X}_R(k)$ is an even function, the inverse discrete Fourier transform of $\hat{X}_R(k)$ gives an even function $c(n)$, which is related to the complex cepstrum $\hat{x}(n)$ by $c(n) = \frac{\hat{x}(n) + \hat{x}(-n)}{2}$.

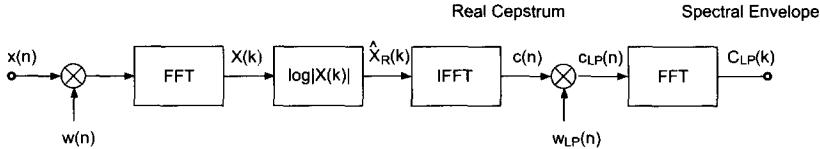


Figure 9.11 Spectral envelope computation by cepstrum analysis.

Figure 9.11 illustrates the computational steps for the computation of the spectral envelope from the real cepstrum. The real cepstrum $c(n)$ is the IFFT of the logarithm of magnitude of FFT of the windowed sequence $x(n)$. The lowpass window for weighting the cepstrum $c(n)$ is derived in [OS75] and is given by

$$w_{LP}(n) = \begin{cases} 1 & n = 0, N_1 \\ 2 & 1 \leq n < N_1 \\ 0 & N_1 < n \leq N - 1. \end{cases} \quad (9.24)$$

with $N_1 \leq N/2$.

The FFT of the windowed cepstrum $c_{LP}(n)$ yields the spectral envelope

$$C_{LP}(k) = \text{FFT}[c_{LP}(n)], \quad (9.25)$$

which is a smoothed version of the spectrum $X(k)$ in dB. An illustrative example is shown in Fig. 9.12. Notice that the first part of the cepstrum $c(n)$ ($0 \leq n \leq 150$) is weighted by the “lowpass window” yielding $c_{LP}(n)$. The IFFT of $c_{LP}(n)$

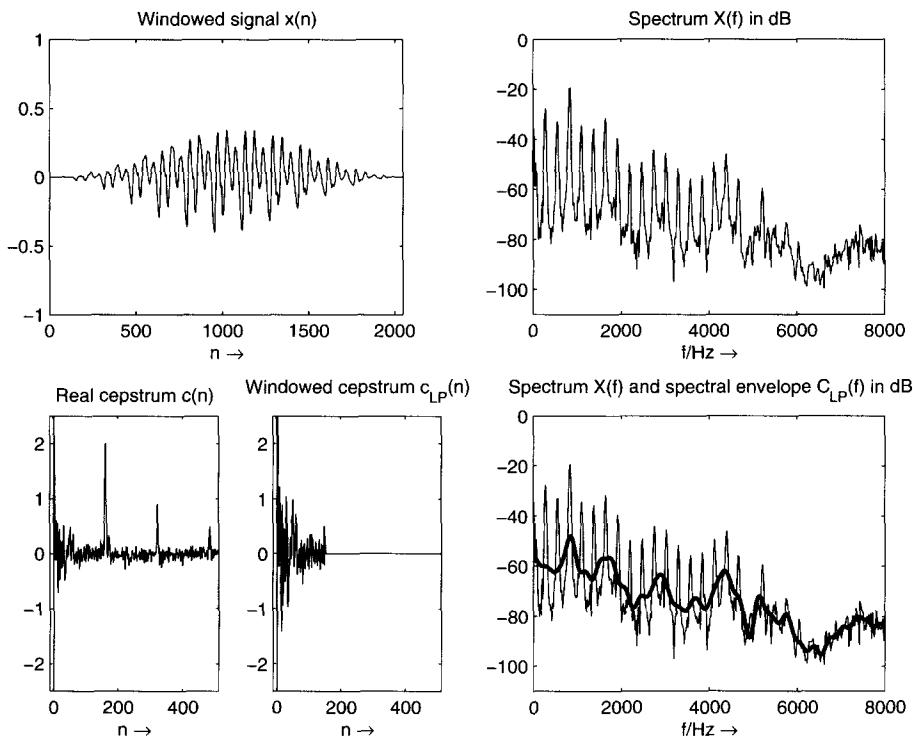


Figure 9.12 Windowed signal segment, spectrum (FFT length $N = 2048$), cepstrum, windowed cepstrum ($N_1 = 150$) and spectral envelope.

results in the spectral envelope $C(f)$ in dB, as shown in the lower right plot. The “highpass part” of the cepstrum $c(n)$ ($150 < n \leq 1024$) represents the source signal, where the first peak at $n = 160$ represents the pitch period T_0 (in samples) of the fundamental frequency $f_0 = 44100 \text{ Hz}/160 = 275,625 \text{ Hz}$. Notice also, although the third harmonic is higher than the fundamental frequency, as can be seen in the spectrum of the segment, the cepstrum method allows the estimation of the pitch period of the fundamental frequency by searching for the time index of the first highly significant peak value in the cepstrum $c(n)$ after the “lowpass” part can be considered to have vanished. The following M-file 9.5 demonstrates briefly the way a spectral envelope can be calculated via the real cepstrum.

```
M-file 9.5 (specenvceps.m)
% N1: cut quefrency
WLen=2048; w=hanningz(WLen);
buf=y(offset:offset+WLen-1).*w;
f=fft(buf)/(WLen/2);
flog=20*log10(0.00001+abs(f));
subplot(2,1,1);plot(flog(1:WLen/2));
```

```
N1=input('cut value for cepstrum');
cep=ifft(flog);
cep_cut=[cep(1);2*cep(2:N1);cep(N1+1);zeros(WLen-N1-1,1)];
flog_cut=real(fft(cep_cut));
subplot(2,1,2);plot(flog_cut(1:WLen/2));
```

In this program `cep` represents the cepstrum (hence the IFFT of the log magnitude of the FFT). The vector `cep_cut` is the version of the cepstrum with all values over the cut index set to zero. Here, we use a programming short-cut: we also remove the negative time values (hence the second part of the FFT) and use only the real part of the inverse FFT. The time indices n of the cepstrum $c(n)$ are also denoted as “quefrequencies”. The vector `flog_cut` is a smoothed version of `flog` and represents the spectral envelope derived by the cepstrum method. The only input value for the spectral envelope computation is the `cut` variable. This variable `cut` is homogeneous to a time in samples, and should be less than the period of the analyzed sound.

Source-filter Separation

The cepstrum method allows the separation of a signal $y(n) = x(n) * h(n)$, which is based on a source and filter model, into its source signal $x(n)$ and its impulse response $h(n)$. The discrete-time Fourier transform $Y(e^{j\Omega}) = X(e^{j\Omega}) \cdot H(e^{j\Omega})$ is the product of two spectra: one representing the filter frequency response $H(e^{j\Omega})$ and the other one the source spectrum $X(e^{j\Omega})$. Decomposing the complex values in terms of the magnitude and phase representation, one can make the strong assumption that the filter frequency response will be real valued and the phase will be assigned to the source signal.

The key point here is to use the mathematical property of the logarithm $\log(a \cdot b) = \log(a) + \log(b)$. The real cepstrum method will perform a spectral envelope estimation based on the magnitude according to

$$\begin{aligned} |Y(e^{j\Omega})| &= |X(e^{j\Omega})| \cdot |H(e^{j\Omega})| \\ \log |Y(e^{j\Omega})| &= \log |X(e^{j\Omega})| + \log |H(e^{j\Omega})|. \end{aligned}$$

In musical terms separating $\log |X(e^{j\Omega})|$ from $\log |H(e^{j\Omega})|$ is to keep the slow variation of $\log |Y(e^{j\Omega})|$ as a filter and the rapid ones as a source. In terms of signal processing we would like to separate the low frequencies of the signal $\log |Y(e^{j\Omega})|$ from its high frequencies (see Fig. 9.13).

The separation of source and filter can be achieved by weighting the cepstrum $c(n) = c_x(n) + c_h(n)$ with two window functions, namely the “lowpass window” $w_{LP}(n)$ and the complementary “highpass window” $w_{HP}(n)$. This weighting yields $c_x(n) = c(n) \cdot w_{HP}(n)$ and $c_h(n) = c(n) \cdot w_{LP}(n)$. The low time values (low “quefrequencies”) for lowpass filtering $\log |Y(e^{j\Omega})|$ give $\log |H(e^{j\Omega})|$ (spectral envelope in dB) and the high time values (higher “quefrequencies”) for highpass filtering yield $\log |X(e^{j\Omega})|$ (source estimation). The calculation of $\exp(\log |H(e^{j\Omega})|)$ gives the magnitude response $|H(e^{j\Omega})|$. From this magnitude transfer function we can compute a

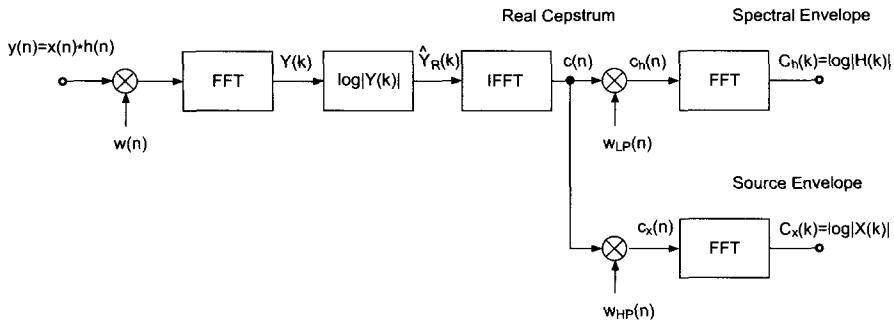


Figure 9.13 Separating source and filter.

zero-phase filter impulse response according to

$$h(n) = \text{IFFT} [|H(k)|]. \quad (9.26)$$

The cepstrum method has a very good by-product. The combination of the highpass filtered version of the cepstrum and the initial phases from the FFT gives a spectrum that can be considered as the source spectrum in a source-filter model. This helps in designing audio effects based on such a source-filter model. The source signal $x(n)$ can be derived from the calculation of $\exp(C_x(k)) = |X(k)|$ and the initial phase taken from $Y(k) = |Y(k)|e^{j\varphi_y(k)}$ by performing the IFFT of $|X(k)|e^{j\varphi_y(k)}$ according to

$$x(n) = \text{IFFT} \left[|X(k)|e^{j\varphi_y(k)} \right]. \quad (9.27)$$

For finding a good threshold between low and high quefrequencies, one can make use of the fact that quefrequencies are time variables, and whenever the sound is periodic, the cepstrum shows a periodicity corresponding to the pitch. Hence this value is the upper quefrency or upper time limit for the spectral envelope. A low value will smoothen the spectral envelope, while a higher value will include some of the harmonic or partial peaks in the spectral envelope.

Hints and Drawbacks

- “Lowpass filtering” is performed by windowing (zeroing values over a “cut quefrency”). This operation corresponds to filtering in the frequency domain with a $\frac{\sin(f)}{f}$ behavior. An alternative version is to use a smooth transition instead of an abrupt cut in the cepstrum domain.
- The cepstrum method will give a spectral estimation that smoothes the instantaneous spectrum. However, log values can go to $-\infty$ for a zero value in the FFT. Though this rarely happens with sounds coming from the real world, where the noise level prevents such values, a good prevention is the limitation of the log value. In our implementation the addition of a small value 0.00001 to the FFT values limits the lower log limit to -100 dB.

- In order to enhance the computation of the spectral envelope, it is possible to use an iterative algorithm which calculates only the positive difference between the instantaneous spectrum and the estimated spectral envelope in each step.
- Though the *real cepstrum* is widely used, it is also possible to use the *complex cepstrum* to perform an estimation of the spectral envelope. In this case the spectral envelope will be defined by a complex FFT.

In conclusion, the cepstrum method allows both the separation of the audio signal into a source signal and a filter and, as a by-product, the estimation of the fundamental frequency, which was already published in [Nol64] and later reported in [Sch99].

9.3 Source-Filter Transformations

9.3.1 Vocoding or Cross-synthesis

The term vocoder has different meanings. One is “voice-coding” and refers directly to speech synthesis. Another meaning for this term is the phase vocoder, which refers to the short-time Fourier transform as discussed in 8.2. The last meaning is the one of the musical instrument named the *Vocoder* and this is what this paragraph is about: vocoding or cross-synthesis.

This effect takes two sound inputs and generates a third one which is a combination of the two input sounds. The general idea is to combine two sounds by “spectrally shaping” the first sound by the second one and preserving the pitch of the first sound. A variant and improvement are the removal of the spectral envelope of the initial sound (also called whitening) before filtering with the spectral envelope of the second one. This implies the ability to extract a spectral envelope evolving with time and to apply it to a signal.

Although spectral estimation is well represented by its amplitude versus frequency representation, most often it is the filter representation that can be a help in the application of this spectral envelope: the channel vocoder uses the weighted sum of filtered bandpass signals, the LPC calculates an IIR filter, and even the cepstrum method can be seen as a circular convolution with an FIR filter. As this vocoding effect is very important and can give different results depending on the technique used, we will introduce these three techniques applied to the vocoding effect.

Channel Vocoder

This technique uses two banks of filters provided by the channel vocoder (see Fig. 9.14), as well as the RMS (root mean square) values associated to these channels. For each channel the bandpass signal is divided by the RMS value of this

channel, and then multiplied by the RMS value of the other sound. The mathematical operation is given by

$$y_{BP_i}(n) = x_{BP_i}(n) \cdot \frac{x_{RMSi_2}(n)}{x_{RMSi_1}(n)}, \quad (9.28)$$

where $x_{RMSi_1}(n)$ and $x_{RMSi_2}(n)$ represent the RMS values in channel i for the two sounds. One should be careful with the division. Of course divisions by zero should be avoided, but there should also be a threshold for avoiding the amplification of noise. This works well when sound 2 has a strong spectral envelope, for example, a voice. The division by $x_{RMSi_1}(n)$ can be omitted or replaced by just modifying the amplitude of each band. Sound 1 can also be a synthetic sound (pulse, sawtooth, square).

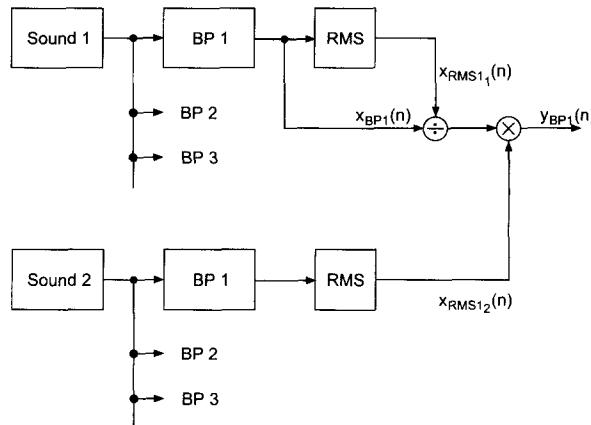


Figure 9.14 Basic principle of spectral mutations.

The following M-file 9.6 demonstrates a cross-synthesis between two sounds based on the channel vocoder implemented by IIR filters.

M-file 9.6 (CVCrossSynthesis.m)

```

%----- USER DATA -----
[y1,FS] = wavread('guitar'); % reading the two sound files
y2 = wavread('xvega');
ly = min(length(y1), length(y2)); % length of the signals
res = zeros(ly,1); % result signal
fen = [0.005 0.0063]; % boundings of frequency band
r = 0.99;
lp = [1, -2*r, +r*r]; % filter used
epsi = 0.00001;
%----- performing the vocoding or cross synthesis effect -----
for k=1:21
    [b, a] = cheby1(2, 3, fen); % chebyshev-type 1 filter

```

```

z1      = filter(b, a, y1);      % filtering the two signals
z2      = filter(b, a, y2);
rms2   = norm(filter(1, lp, z2.*z2),2);% RMS value of sound 2
% rms1   = epsi + norm(filter(1, lp, z1.*z1), 2);% with whitening
rms1   = 1.;                      % without whitening
res    = res + z1.*rms2/rms1;    % add result to the output buffer
fen    = fen*1.26;                % width of the bandpass filter:
                                  % 1/3 of an octave = 2^(1/3)^1.26
end
soundsc(res,FS)

```

This program performs bandpass filtering inside a loop. Precisely, Chebychev type 1 filters are used, which are IIR filters with a ripple of 3 dB in the passband. The bandwidth is chosen as one-third of an octave, hence the 0.005 to 0.0063 window relative to half of the sampling rate in Matlab's definition. Then sound 1 and sound 2 are filtered, and the RMS value of the filtered sound 2 is extracted: z2 is squared, filtered by a two pole filter on the x axis, and its square root is taken. This RMS2 value serves as a magnitude amplifier for the z1 signal, which is the filtered version of sound 1. This operation is repeated every one-third of an octave by multiplying the frequency window, which is used for the definition of the filter, by 1.26 (3rd root of 2). A whitening process can be introduced by replacing line `rms1 = 1.;` with `rms1 = epsi + norm(filter(1, lp, z1.*z1), 2);`. A small value `epsi` (0.01) is added to RMS1 to avoid division by zero. If `epsi` is greater, the whitening process is attenuated. Thus this value can be used as a control for the whitening.

Linear Prediction

Cross-synthesis between two sounds can also be performed using the LPC method [Moo79, KAZ00]. One filter removes the spectral envelope of the first sound and the spectral envelope of the second sound is used to filter the excitation signal of the first sound, as shown in Fig. 9.15.

The following M-file 9.7 performs cross-synthesis based on the LPC method. The prediction coefficients of sound 1 are used for an FIR filter to whiten the original sound. The prediction coefficients of sound 2 are used in the feedback path of a synthesis filter, which performs filtering of the excitation signal of sound 1 with the spectral envelope derived from sound 2.

```

M-file 9.7 (LPCCrossSynthesis.m)
===== LPCCrossSynthesis.m =====
clear; clf;

----- USER DATA -----
[DAFx_in1,FS] = wavread('didge_court.wav'); % sound 1: spectral env.
DAFx_in2 = wavread('song.wav');             % sound 2: excitation
long     = 400;    % block length for calculation of coefficients
hopsize  = 160;    % hop size (is 160)

```

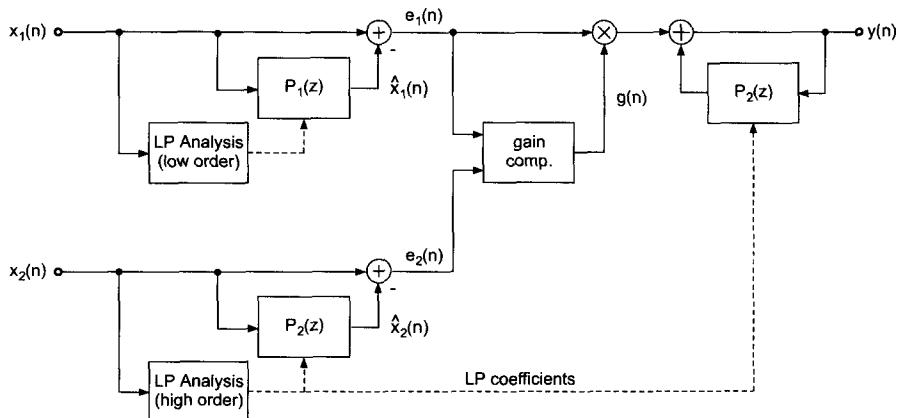


Figure 9.15 Cross-synthesis with LPC.

```

order      = 20          % order of the LPC
order1     = 6           % order for the excitation

%----- initializations -----
ly        = min(length(DAFx_in1), length(DAFx_in2));
DAFx_in1 = [zeros(order, 1); DAFx_in1; ...
            zeros(order-mod(ly,hopsize),1)] / max(abs(DAFx_in1));
DAFx_in2 = [zeros(order, 1); DAFx_in2; ...
            zeros(order-mod(ly,hopsize),1)] / max(abs(DAFx_in2));
DAFx_out = zeros(ly,1);                                % result sound
exc      = zeros(ly,1);                                % excitation sound
w        = hanningz(long);                            % window
N_frames = floor((ly-order-long)/hopsize); % number of frames

%----- Cross-synthesis -----
tic
for j=1:N_frames
    k        = order + hopsize*(j-1);      % offset of the buffer
    [A, g]   = lpc(DAFx_in2(k+1:k+long).*w, order);
    [A1, g1] = lpc(DAFx_in1(k+1:k+long).*w, order1);

    gain(j)  = g;                         %
    ae       = - A(2:order+1);             % LPC coeff. of excitation
    for n=1:hopsize
        excitation1 = (A1/g1) * DAFx_in1(k+n:-1:k+n-order1);
        exc(k+n)    = excitation1;
        DAFx_out(k+n) = ae*DAFx_out(k+n-1:-1:k+n-order)+g*excitation1;
    end
end

```

```

toc
%----- output -----
DAFx_out = DAFx_out(order+1:length(DAFx_out)) / max(abs(DAFx_out));
soundsc(DAFx_out, FS)
wavwrite(DAFx_out, FS, 'CrossLPC')

```

Cepstrum

Signal processing based on cepstrum analysis is also called homomorphic signal processing [OS75, PM96]. We have seen that we can derive the spectral envelope (in dB) with the cepstrum technique. Reshaping a sound is achieved by whitening (filtering) a sound with the inverse spectral envelope $1/|H_1(f)|$ and then filtering with the spectral envelope $|H_2(f)|$ of the second sound (see Fig. 9.16). The series connection of both filters leads to a transfer function $H_2(f)/H_1(f)$. By taking the logarithm according to $\log |H_2(f)|/|H_1(f)| = \log |H_2(f)| - \log |H_1(f)|$, the filtering operation is based on the difference of the two spectral envelopes. The first spectral envelope performs the whitening by inverse filtering and the second spectral envelope introduces the formants. The inverse filtering of the input sound 1 and subsequent filtering with spectral envelope of sound 2 can be performed in one step by the fast convolution technique.

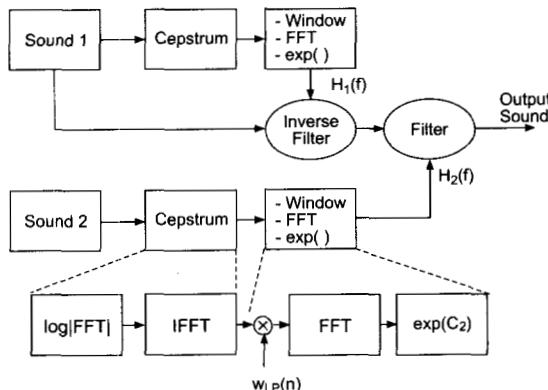


Figure 9.16 Basic principle of homomorphic cross-synthesis. The spectral envelopes of both sounds are derived by the cepstrum method.

Here we present the core of a program given by M-file 9.8 that uses the spectral envelope of a sound (number 2) to be superimposed on a sound (number 1). Though musically very effective, this first program does not do any whitening of sound 1.

M-file 9.8 (CepstrumCrossSynthesis.m)

```

% CepstrumCrossSynthesis.m
clear all; close all
%----- USER DATA -----

```

```

[DAFx_in1, FS] = wavread('didge_court.wav'); % sound 1: excitation
DAFx_in2 = wavread('la.wav'); % sound 2: spectral envelope
WLen = 1024; % window size
n1 = 256; % hop size
order1 = 30; % cut quefrency for sound 1
order2 = 30; % cut quefrency for sound 2

%----- initializations -----
w1 = hanningz(WLen); % analysis window
w2 = w1; % synthesis window
WLen2 = WLen/2
grain1 = zeros(WLen,1);
grain2 = zeros(WLen,1);
pin = 0; % start index
L = min(length(DAFx_in1),length(DAFx_in2));
pend = L - WLen; % end index
DAFx_in1 = [zeros(WLen, 1); DAFx_in1; ...
    zeros(WLen-mod(L,n1),1)] / max(abs(DAFx_in1));
DAFx_in2 = [zeros(WLen, 1); DAFx_in2; ...
    zeros(WLen-mod(L,n1),1)] / max(abs(DAFx_in2));
DAFx_out = zeros(L,1);

%----- cross-synthesis -----
while pin<pend
    grain1 = DAFx_in1(pin+1:pin+WLen).* w1;
    grain2 = DAFx_in2(pin+1:pin+WLen).* w1;
%=====
    f1 = fft(grain1);

    f = fft(grain2)/WLen2;
    flog = log(0.00001+abs(f));
    cep = ifft(flog); % cepstrum of sound 2
    cep_couple = [cep(1)/2; cep(2:order1); zeros(WLen-order1,1)];
    flog_couple = 2*real(fft(cep_couple));
    f2 = exp(flog_couple); % spectral env. of sound 2

    grain = (real(ifft(f1.*f2))).*w2;% resynthesis grain
% =====
    DAFx_out(pin+1:pin+WLen) = DAFx_out(pin+1:pin+WLen)+grain;
    pin = pin + n1;
end

%----- listening and saving the output -----
%DAFx_in = DAFx_in(WLen+1:WLen+L);
DAFx_out = DAFx_out(WLen+1:length(DAFx_out))/max(abs(DAFx_out));

```

```
soundsc(DAFx_out,FS);
wavwrite(DAFx_out,FS,'CrossCepstrum')
```

In this program `n1` represents a hop size, and `grain1` and `grain2` windowed buffers of `sound1` and `sound2`. `f1` is the FFT of `grain1` and `f2` is the spectral envelope derived from the FFT of `grain2`. Although this algorithm performs a circular convolution, which theoretically introduces time aliasing, the resulting sound does not have artifacts.

Whitening `sound1` before processing it with the spectral envelope of `sound2` can be done in a combined step: we calculate the spectral envelope of `sound1` and subtract it (in dB) from the spectral envelope of `sound2`. The following code lines given by M-file 9.9 perform a whitening of `sound1` and a cross-synthesis with `sound2`.

```
M-file 9.9 (CepstrumWhiteningCS.m)
%=====
f1      = fft(grain1)/WLen2;
flog    = log(0.00001+abs(f1));
cep     = flog;                      % cepstrum of sound 1
cep_coupe = [cep(1)/2; cep(2:order1); zeros(WLen-order1,1)];
flog_couple1 = 2*real(ifft(cep_coupe)); % spectral env. of sound 2

f2      = fft(grain2)/WLen2;
flog    = log(0.00001+abs(f2));
cep     = ifft(flog);                % cepstrum of sound 2
cep_coupe = [cep(1)/2; cep(2:order2); zeros(WLen-order2,1)];
flog_couple2 = 2*real(fft(cep_coupe)); % spectral env. of sound 2

f2      = exp(flog_couple2-flog_couple1); % whitening
                                              % the excitation
grain   = (real(ifft(f1.*f2))).*w2;       % resynthesis grain
% =====
```

In this program `flogcoupe1` and `flogcoupe2` represent (in dB) the spectral envelopes derived from `grain1` and `grain2` for a predefined cut quefrency. Recall that this value is given in samples. It should normally be below the pitch period of the sound, and the lower it is, the more smoothed the spectral envelope will be.

9.3.2 Formant Changing

This effect produces a “Donald Duck” voice without any alteration of the fundamental frequency. It can be used for performing an alteration of a sound whenever there is a formant structure. However, it can also be used in conjunction with pitch shifted sounds for recovering a natural formant structure (see section 9.3.4).

The musical goal is to remove the spectral envelope from one sound and to impose another one, which is a warped version of the first one, as shown in Fig. 9.17,

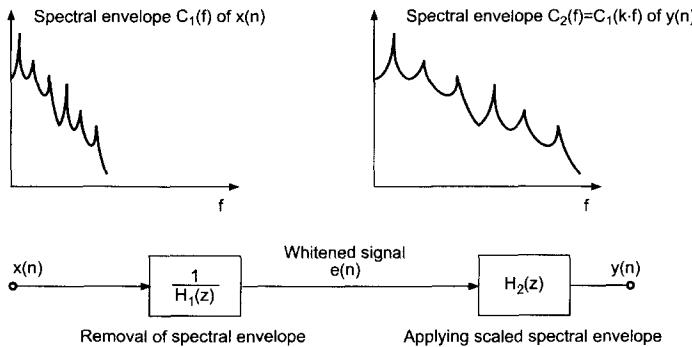


Figure 9.17 Formant changing by frequency scaling the spectral envelope and time-domain processing.

where the signal processing is also illustrated. This means that we have to use a spectral correction that is a ratio of the two spectral envelopes. In this way the formants, if there are any, are changed according to this warping function. For example, a transposition of the spectral envelope by a factor of two will give a “Donald Duck” effect without time stretching. This effect can also be seen as a particular case of cross-synthesis, where the modifier comes from an interpolated version of the original sound. Though transposition of the spectral envelope is classical, other warping functions can be used.

From a signal processing point of view the spectral correction for formant changing can be seen in the frequency domain as $H_2(f)/H_1(f)$. First divide by the spectral envelope $H_1(f)$ of the input sound and then multiply by the frequency scaled spectral envelope $H_2(f)$. In the cepstrum domain the operation $H_2(f)/H_1(f)$ leads to the subtraction $C_2(f) - C_1(f)$, where $C(f) = \log|H(f)|$. When using filters for time-domain processing, the transfer function is $H_2(f)/H_1(f)$ (see Fig. 9.17). We will shortly describe three different methods for the estimation of the two spectral envelopes.

Interpolation of the Input Signal

The spectral envelopes $C_1(f)$ and $C_2(f)$, or filters $H_1(f)$ and $H_2(f)$ can be obtained by different techniques. If $C_2(f)$ is a frequency scaled version of $C_1(f)$, one can calculate the spectral envelope $C_2(f)$ from the analysis of a transposed version of the initial signal, as shown in Fig. 9.18. The transposed version is obtained by time-domain interpolation of the input signal. The channel vocoder, LPC and the cepstrum method allow the estimation of either the spectral envelope or the corresponding filter. One must take care to keep synchronicity between the two signals. This can be achieved by changing the hop size according to this ratio. The algorithm works as follows:

- Whitening: filter the input signal with frequency response $\frac{1}{H_1(f)}$ or subtract

the input spectral envelope $C_1(f) = \log |H_1(f)|$ from the log of input magnitude spectrum.

- The filter $H_1(f)$ or the spectral envelope $C_1(f)$ is estimated from the input signal.
- Formant changing: apply the filter with frequency response $H_2(f)$ to the whitened signal or add the spectral envelope $C_2(f) = \log |H_2(f)|$ to the whitened log of the input magnitude spectrum.
- The filter $H_2(f)$ or the spectral envelope $C_2(f)$ is estimated from the interpolated input signal.

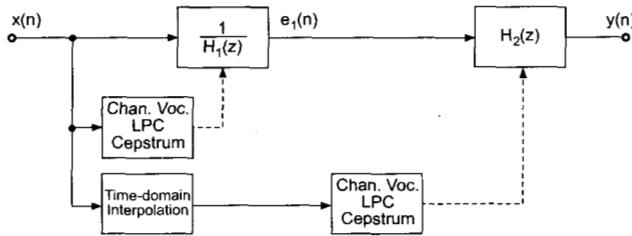


Figure 9.18 Formant changing by time-domain processing.

Formant changing based on the cepstrum analysis is shown in Fig. 9.19. The spectral correction is calculated from the difference of the log values of the FFTs, of both the input signal and the interpolated input signal. This log difference is transformed to the cepstrum domain, lowpass weighted and transformed back by the exponential to the frequency domain. Then, the filtering of the input signal with the spectral correction filter $H_2(z)/H_1(z)$ is performed in the frequency domain. This fast convolution is achieved by multiplication of the corresponding Fourier transforms of the input signal and the spectral correction filter. The result is transformed back to the time domain by an IFFT yielding the output signal. An illustrative example is shown in Fig. 9.20. The M-file 9.10 demonstrates this technique.

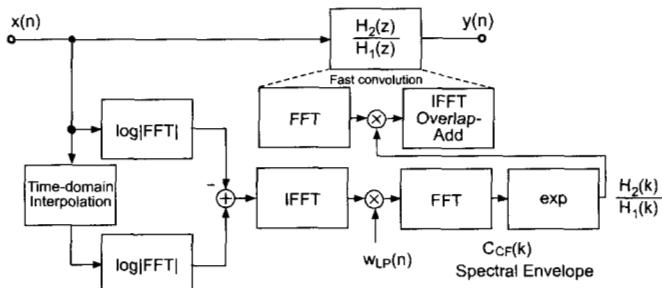


Figure 9.19 Formant changing by frequency-domain processing: cepstrum analysis, spectral correction filter computation and fast convolution.

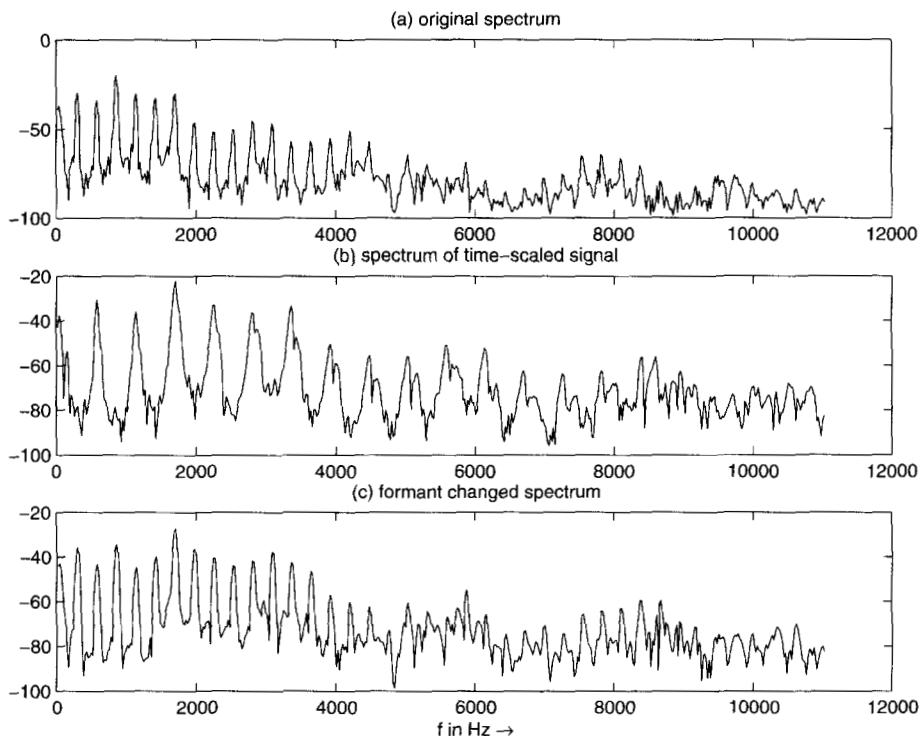


Figure 9.20 Example of formant changing: the upper plot shows the input spectrum and the middle plot the spectrum of the interpolated signal. The lower plot shows the result of the formant changing operation, where the spectral envelope of the interpolated spectrum can be noticed.

M-file 9.10 (UX_fmove_cepstrum.m)

```
% UX_fmove_cepstrum.m
% UX_fmove_cepstrum.m
clear; clf;

%----- USER DATA -----
[DAFx_in, FS] = wavread('la.wav'); % sound file
warping_coef = 2.0;
n1 = 512; % analysis hop size
n2 = n1; % synthesis hop size
WLen = 2048; % window length
w1 = hanningz(WLen); % analysis window
w2 = w1; % synthesis window
order = 50; % cut quefrency

%----- initializations -----
WLen2 = WLen/2;
```

```

L           = length(DAFx_in);
DAFx_in     = [zeros(WLen, 1); DAFx_in; ...;
               zeros(WLen-mod(L,n1),1)] / max(abs(DAFx_in));
DAFx_out    = zeros(L,1);
t           = 1 + floor((0:WLen-1)*warping_coef); % warping
lmax        = max(WLen,t(WLen))

tic
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
pin = 0;
pout = 0;
pend = length(DAFx_in)-lmax;
%pin = 6500;           % for plot
%pend = pin + WLen; % for plot
while pin<pend
  grain = DAFx_in(pin+1:pin+WLen).* w1;
%=====
  f      = fft(grain)/WLen2;          % spectrum of grain
  flogs = 20*log10(0.00001+abs(f));% log|X(k)|

  grain1 = DAFx_in(pin+t).* w1;      % linear interpolation of grain
  f1     = fft(grain1)/WLen2;        % spectrum of interp. grain
  flogs1 = 20*log10(0.00001 + abs(f1));% log|X1(k)|
  flog   = log(0.00001+abs(f1)) - log(0.00001+abs(f));
  cep    = ifft(flog);             % cepstrum
  cep_couple = [cep(1)/2; cep(2:order); zeros(WLen-order,1)];

  corr   = exp(2*real(fft(cep_couple)));% spectral envelope
  grain  = (real(ifft(f.*corr))).*w2;

  fout   = fft(grain);
  flogs2 = 20*log10(0.00001+abs(fout));

  %---- figures for real-time spectral envelope up to FS/2 ----
  subplot(3,1,1);plot((1:WLen2/2)*44100/WLen,flogs(1:WLen2/2));
  title('a) original spectrum'); drawnow;
  subplot(3,1,2);plot((1:WLen2/2)*44100/WLen,flogs1(1:WLen2/2));
  title('b) spectrum of time-scaled signal');
  subplot(3,1,3);plot((1:WLen2/2)*44100/WLen,flogs2(1:WLen2/2));
  title('c) formant changed spectrum');
  xlabel('f in Hz \rightarrow');
  drawnow
% =====
  DAFx_out(pout+1:pout+WLen)=DAFx_out(pout+1:pout+WLen)+grain;
  pin = pin + n1;
  pout = pout + n2;

```

Interpolation or Scaling of the Spectral Envelope

The direct warping is also possible, for example, by using the interpolation of the spectral envelope derived from a cepstrum technique: $C_2(f) = C_1(k \cdot f)$ or $C_2(f/k) = C_1(f)$. There are, however, numerical limits: the cepstrum method uses an FFT and frequencies should be below half of the sampling frequency. Thus, if the transposition factor is greater than one, we will get only a part of the initial envelope. If the transposition factor is less than one, we will have to zero-pad the rest of the spectral envelope to go up to half of the sampling frequency. The block diagram for the algorithm using the cepstrum analysis method is shown in Fig. 9.21. The following M-file 9.11 demonstrates this method.

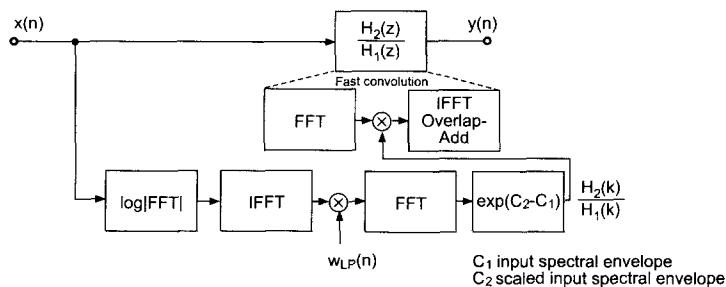


Figure 9.21 Formant changing by scaling the spectral envelope.

M-file 9.11 (UX_fomove_cepstrum.m)

```
% UX_fomove_cepstrum.m
clear; clf;

%----- USER DATA -----
[DAFx_in, FS] = wavread('la.wav'); % sound file
warping_coef = 2.;
n1 = 512; % analysis hop size
n2 = n1; % synthesis hop size
WLen = 2048; % window length
```

```

w1           = hanningz(WLen);      % analysis window
w2           = w1;                  % synthesis window
order        = 50;                 % cut quefrency

%----- initializations -----
WLen2 = WLen / 2;
L          = length(DAFx_in);
DAFx_in   = [zeros(WLen, 1); DAFx_in; ...
             zeros(WLen-mod(L,n1),1)] / max(abs(DAFx_in));
DAFx_out  = zeros(L,1);
x0         = floor(min((1+(0:WLen2)/warping_coef), 1+WLen2));
            % apply the warping
x          = [x0, x0(WLen2:-1:2)];% symmetric extension

tic
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
pin = 0;
pout = 0;
pend = L - WLen;

while pin<pend
    grain = DAFx_in(pin+1:pin+WLen).* w1;
%=====
    f      = fft(grain)/WLen2;
    flog  = log(0.00001+abs(f));
    cep   = ifft(flog);
    cep_couple = [cep(1)/2; cep(2:order); zeros(WLen-order,1)];
    % flogcoupe 1/2 are the spectral envelopes
    % before/after formant move
    flog_couple1 = 2*real(fft(cep_couple));
    flog_couple2 = flog_couple1(x);
    corr = exp(flog_couple2-flog_couple1);
    grain = (real(ifft(f.*corr))).*w2;
%=====
    DAFx_out(pout+1:pout+WLen) = DAFx_out(pout+1:pout+WLen) + grain;
    pin   = pin + n1;
    pout  = pout + n2;
end
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
toc

%----- listening and saving the output -----
%DAFx_in = DAFx_in(WLen+1:WLen+L);
DAFx_out = DAFx_out(WLen+1:WLen+L) / max(abs(DAFx_out));
soundsc(DAFx_out, FS);
wavwrite(DAFx_out, FS, 'la_move.wav');

```

Direct Warping of Filters

A direct warping of the spectral envelope filter $H_1(z)$ to $H_2(z)$ is also possible. The warping of a filter transfer function can be performed by the allpass function $z^{-1} \rightarrow \frac{z^{-1}-\alpha}{1-\alpha z^{-1}}$. Substituting z^{-1} in the transfer function $H_1(z)$ by $\frac{z^{-1}-\alpha}{1-\alpha z^{-1}}$ yields the warped transfer function $H_2(z)$. Further details on warping can be found in Chapter 11 and in [Str80, LS81, HKS⁺00].

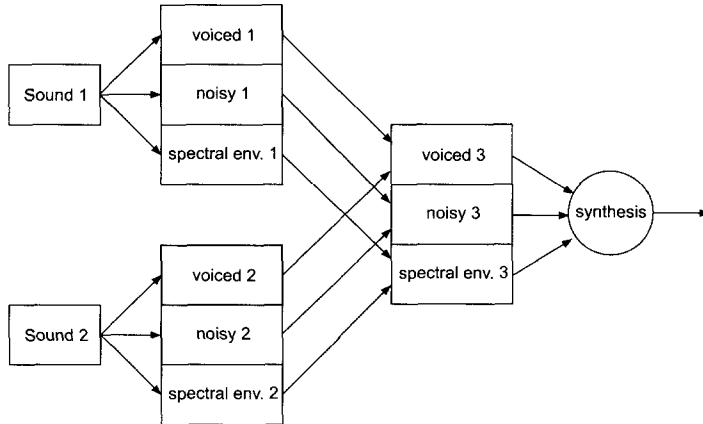


Figure 9.22 A possible implementation of spectral interpolation between two sounds.

9.3.3 Spectral Interpolation

Spectral interpolation means that instead of mixing two sounds, we mix their excitation signals independently of their spectral envelopes, as shown in Fig. 9.22. If we have the decomposition of sound grains in the frequency domain according to $E(f) \cdot H(f)$, where $E(f)$ represents the Fourier transform of the excitation and $H(f)$ is the spectral envelope ($H(f) = \exp[C(f)]$), we can perform spectral interpolation between two sounds by mixing according to

$$Y(f) = [e_1 E_1(f) + e_2 E_2(f)] \cdot [c_1 H_1(f) + c_2 H_2(f)]. \quad (9.29)$$

The excitation grains and the spectral envelopes are added. This transformed representation is then used for the resynthesis operation. We introduce cross-terms by this method, which musically means that the excitation source of one sound also influences the spectral envelope of the second and conversely. For regular mixing of two sounds the result would be $k_1 E_1(f) \cdot H_1(f) + k_2 E_2(f) \cdot H_2(f)$. The M-file 9.12 performs time-varying spectral interpolation between two sounds. We go from a first sound to another one by independently mixing the sources and resonances of these two sounds.

M-file 9.12 (UX_spectral_interp.m)
`% UX_spectral_interp.m`

```

% k (spectral mix)
% =0 ->x1 =1->x2 in between spectral interpolation
% in this example k is calculated at every step
% so we move from sound 1 to sound 2;

clear; clf

%---- USER DATA -----
[DAFx_in1,FS] = wavread('claire_oubli_voix.WAV'); % sound 1
DAFx_in2 = wavread('claire_oubli_flute.WAV'); % sound 2
n1 = 512; % analysis hop size
n2 = n1; % synthesis hop size
WLen = 2048; % window length
w1 = hanningz(WLen); % analysis window
w2 = w1; % synthesis window
cut = 50 % cut quefrency

%---- initializations -----
L = min(length(DAFx_in1), length(DAFx_in2));
DAFx_in1 = [zeros(WLen, 1); DAFx_in1; ...
    zeros(WLen-mod(L,n1),1)] / max(abs(DAFx_in1));
DAFx_in2 = [zeros(WLen, 1); DAFx_in2; ...
    zeros(WLen-mod(L,n1),1)] / max(abs(DAFx_in2));
DAFx_out = zeros(length(DAFx_in1),1);

tic
%%%%%%%%%%%%%
pin = 0;
pout = 0;
pend = L - WLen;
while pin<pend
    %---- here is the k factor varies between 0 and 1
    k = pin/pend; % spectral mix
    kp = 1-k;

    grain1 = DAFx_in1(pin+1:pin+WLen).* w1;
    grain2 = DAFx_in2(pin+1:pin+WLen).* w1;
%=====
    f1 = fft(fftshift(grain1));
    flog = log(0.00001+abs(f1));
    cep = fft(flog);
    cep_cut = [cep(1)/2; cep(2:cut); zeros(WLen-cut,1)];
    flog_cut1 = 2*real(ifft(cep_cut));
    spec1 = exp(flog_cut1); % spectral shape of sound 1

```

```

f2      = fft(fftshift(grain2));
flog   = log(0.00001+abs(f2));
cep    = fft(flog);
cep_cut  = [cep(1)/2; cep(2:cut); zeros(WLen-cut,1)];
flog_cut2 = 2*real(ifft(cep_cut));
spec2 = exp(flog_cut2);           % spectral shape of sound 2

%---- here we interpolate the spectral envelopes in dB
spec  = exp(kp*flog_cut1+k*flog_cut2);

ft     = (kp*f1./spec1+k*f2./spec2).*spec;
grain = fftshift(real(ifft(ft))).*w2;
=====
DAFx_out(pout+1:pout+WLen)=DAFx_out(pout+1:pout+WLen)+grain;
pin  = pin + n1;
pout = pout + n2;
end
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
toc

%---- listening and saving the output ----
%DAFx_in = DAFx_in1(WLen+1:WLen+L);
DAFx_out = DAFx_out(WLen+1:WLen+L) / max(abs(DAFx_out));
soundsc(DAFx_out, FS);
wavwrite(DAFx_out, FS, 'spec_interp.wav');

```

Spectral interpolation is the first step towards morphing, a term borrowed from the visual domain but with a much more ambiguous meaning in the audio domain. Time synchronization between the two sounds has to be taken into account. The matter of pitch interpolation should be different from the mixing of excitation signals as it is presented here. Morphing usually relies on high level attributes and spectral interpolation and follows more complicated schemes, such as shown in Fig. 9.22. Advanced methods will be discussed in Chapter 10.

9.3.4 Pitch Shifting with Formant Preservation

In Chapter 7, we saw some pitch shifting algorithms which transpose the entire spectrum, and consequently the spectral envelope. This typically alters the voice giving a “Donald Duck” or “barrel” feeling. For pitch shifting a sound without changing its articulation, i.e. its formant structure, one has to keep the spectral envelope of the original sound.

Inverse Formant Move plus Pitch Shifting

A possible way to remove the artifacts is to perform a formant move in the inverse direction of the pitch shifting. This process can be inserted into a FFT/IFFT based

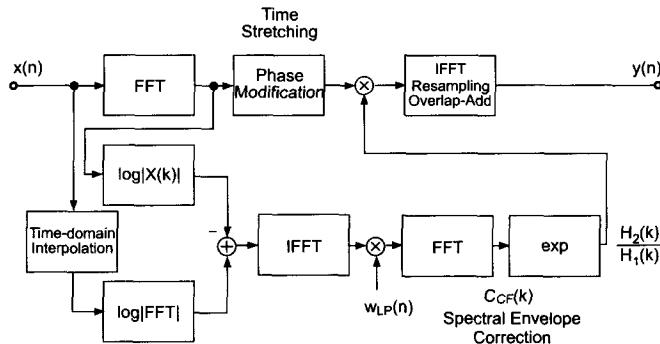


Figure 9.23 Pitch shifting with formant preservation: the pitch shifting is performed in the frequency domain.

pitch shifting algorithm before the reconstruction, as shown in Fig. 9.23. For this purpose we have to calculate a correction function for this formant move.

The following algorithm (see M-file 9.13) is based on the pitch shifting algorithm described in Chapter 8 (see section 8.4.4). The only modification is a formant move calculation before the reconstruction of every individual grain which will be overlapped and added. For the formant move calculation, a crude interpolation of the analysis grain is performed, in order to recover two spectral envelopes: the one of the original grain and the one of its pitch-transposed version. From these two spectral envelopes the correction factor is computed (see previous section) and applied to the magnitude spectrum of the input signal before the reconstruction of the output grain (see Fig. 9.23).

```
M-file 9.13 (UX_pitch_pv_move.m)
% UX_pitch_pv_move.m
clear; clf

%----- USER DATA -----
[DAFx_in, FS] = wavread('la.wav'); % sound file
n1          = 512;   % analysis hop size
                  % try n1=400 (pitch down) or 150 (pitch up)
n2          = 256;   % synthesis hop size
                  % keep it a divisor of WLen (256 is good)
WLen        = 2048;  % window length
w1          = hanningz(WLen); % analysis window
w2          = w1;    % synthesis window
order       = 50;    % cut quefrency

%----- initializations -----
ral          = n2/n1;
WLen2 = WLen/2;
% for linear interpolation of a grain of length WLen
```

```

lx          = floor(WLen*n1/n2);
x           = 1 + (0:lx-1)'*WLen/lx;
ix          = floor(x);
ix1         = ix + 1;
dx          = x - ix;
dx1         = 1 - dx;
warping_coef = n1/n2
t           = 1 + floor((0:WLen-1)*warping_coef);
lmax        = max(WLen,t(WLen))
L           = length(DAFx_in);
DAFx_in    = [zeros(WLen, 1); DAFx_in; ...
             zeros(WLen-mod(L,n1),1)] / max(abs(DAFx_in));
DAFx_out   = zeros(lx+length(DAFx_in),1);
omega       = 2*pi*n1*[0:WLen-1]'/WLen;
phi0        = zeros(WLen,1);
psi         = zeros(WLen,1);

tic
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
pin        = 0;
pout       = 0;
pend       = L - lmax;

while pin<pend
    grain   = DAFx_in(pin+1:pin+WLen).* w1;
%=====
    f       = fft(fftshift(grain));
    r       = abs(f);
    phi    = angle(f);

    delta_phi = omega + princarg(phi-phi0-omega); % phase unwrapping
    phi0    = phi;
    psi     = princarg(psi+delta_phi*r1);

    %---- formant move ----
    grain1 = DAFx_in(pin+t) .* w1;
    f1     = fft(grain1)/WLen2;
    flog   = log(0.00001+abs(f1))-log(0.00001+abs(f));
    cep    = ifft(flog);
    cep_cut = [cep(1)/2; cep(2:order); zeros(WLen-order,1)];
    corr   = exp(2*real(fft(cep_cut)));

    ft     = (r.* corr.* exp(i*psi));
    grain  = fftshift(real(ifft(ft))).*w2;

    %---- interpolation ----

```

```

grain2 = [grain;0];
grain3 = grain2(ix).*dx1+grain2(ix1).*dx;
% plot(grain);drawnow;
=====
DAFx_out(pout+1:pout+lx) = DAFx_out(pout+1:pout+lx) + grain3;
pin    = pin + n1;
pout   = pout + n1;
end
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
toc

----- listening and saving the output -----
DAFx_in = DAFx_in(WLen+1:WLen+L);
DAFx_out = DAFx_out(WLen+1:WLen+L) / max(abs(DAFx_out));
soundsc(DAFx_out, FS);
wavwrite(DAFx_out, FS, 'x1_pitch_pv_move.wav');

```

An illustrative example of pitch shifting with formant preservation is shown in Fig. 9.24. The spectral envelope is preserved and the pitch is increased by a factor of two.

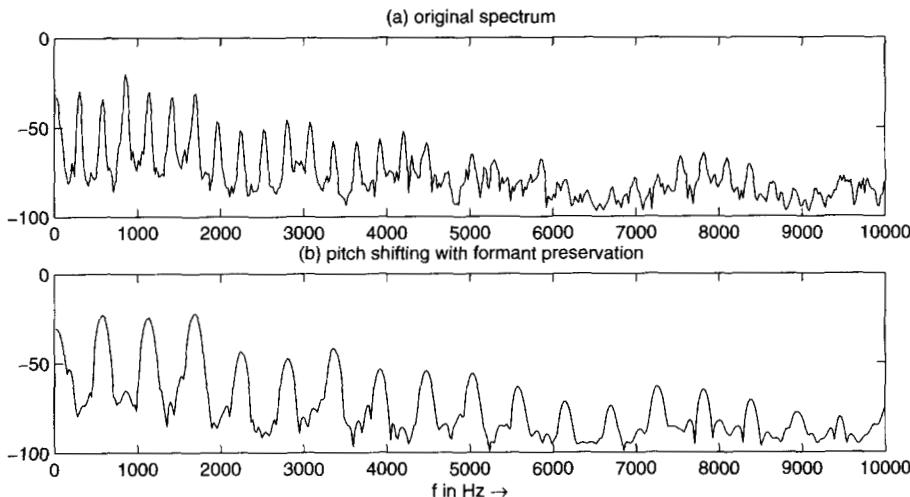


Figure 9.24 Example of pitch shifting with formant preservation.

Resampling plus Formant Move

It is also possible to combine an interpolation scheme with a formant move inside an analysis-synthesis loop. The block diagram in Fig. 9.25 demonstrates this approach. The input segments are interpolated from length N_1 to length N_2 . This interpolation

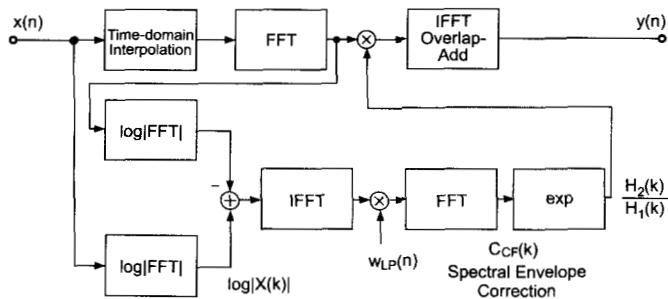


Figure 9.25 Pitch shifting with formant preservation: the pitch shifting is performed in the time domain.

or resampling also changes the time duration and thus performs pitch shifting in the time domain. The resampled segment is then applied to an FFT/IFFT-based analysis/synthesis system, where the correction function for the formant move is computed by the cepstrum method. This correction function is based on the input spectrum and the spectrum of the interpolated signal and is computed with the help of the cepstrum technique. Then the correction function is applied to the interpolated input spectrum by the fast convolution technique. The following M-file 9.14 performs interpolation of successive grains with a ratio given by the two numbers $R_a = n1$ and $R_s = n2$ and performs a formant move to recover the original spectral envelope.

```

M-file 9.14 (UX_interp_move.m)
% UX_interp_move.m
clear; clf

%----- USER DATA -----
[DAFx_in, FS] = wavread('la.wav'); % sound file
n1 = 400; % analysis hop size
            % try n1=400 (pitch down) or 150 (pitch up)
n2 = 256; % synthesis hop size
            % keep it a divisor of WLen (256 is good)
WLen = 2048; % window length
w1 = hanningz(WLen); % analysis window
w2 = w1; % synthesis window
order = 50; % cut quefrency

%----- initializations -----
ral = n2/n1
WLen2 = WLen/2;
% for linear interpolation of a grain of length WLen
lx = floor(WLen*n1/n2);
x = 1 + (0:WLen-1)'*lx/WLen;
  
```

```

ix          = floor(x);
ix1         = ix + 1;
dx          = x - ix;
dx1         = 1 - dx;
warping_coef = n1/n2;
lmax         = max(WLen,1x)
L            = length(DAFx_in);
DAFx_in     = [zeros(WLen, 1); DAFx_in] / max(abs(DAFx_in));
DAFx_out    = zeros(ceil(ral*length(DAFx_in)),1);

tic
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
pin = 0;
pout = 0;
pend = L - lmax;

while pin<pend
%=====
%---- interpolated grain
grain1 = (DAFx_in(pin+ix).*dx1 + DAFx_in(pin+ix1).*dx).* w1;
f1    = fft(grain1)/WLen2;
%---- reference grain for formant matching
grain2 = DAFx_in(pin+1:pin+WLen).* w1;
f2    = fft(grain2)/WLen2;
%---- correcting factor for spectral envelope
flog  = log(0.00001+abs(f2))-log(0.00001+abs(f1));
cep   = ifft(flog);
cep_cut = [cep(1)/2; cep(2:order); zeros(WLen-order,1)];
corr  = exp(2*real(fft(cep_cut)));
%---- so now make the formant move
grain = fftshift(real(ifft(f1.*corr))).*w2;
% plot(grain);drawnow;
%=====

DAFx_out(pout+1:pout+WLen) = DAFx_out(pout+1:pout+WLen) + grain;
pin   = pin + n1;
pout  = pout + n2;
end
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
toc

%---- listening and saving the output -----
DAFx_in = DAFx_in(WLen+1:WLen+L);
DAFx_out = DAFx_out(WLen+1:length(DAFx_out)) / max(abs(DAFx_out));
soundsc(DAFx_out, FS);
wavwrite(DAFx_out, FS, 'x1_interp_move.wav');

```

Resampling of the Excitation Signal

Instead of moving the formants, an alternative technique is to calculate an excitation signal by removing the spectral envelope, to process the excitation signal by a pitch shifting algorithm and to filter the pitch shifted excitation signal with the original spectral envelope. LPC algorithms can be used for this approach. Figure 9.26 shows a block diagram of pitch shifting with formant preservation using the LPC method. First, a predictor is computed and the predictor is used for the inverse filtering of the input signal, which yields the excitation signal. Then the excitation signal is applied to a pitch shifting algorithm and the output signal is filtered with the synthesis filter $H_1(z)$. The processing steps can be performed completely in the time domain. The pitch shifting is achieved by first time stretching and subsequent resampling.

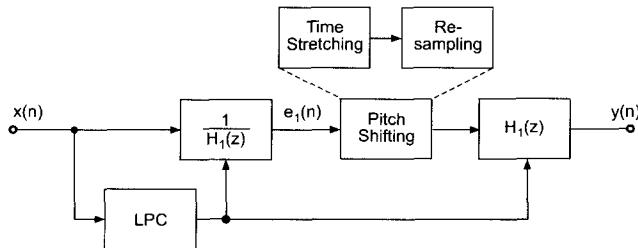


Figure 9.26 Pitch shifting with formant preservation with the LPC method.

9.4 Feature Extraction

A musical sound has some perceptive features that can be extracted from a time-frequency representation. As an example, pitch is a function of time that is very important for musicians, but richness of timbre, inharmonicity, balance between odd and even harmonics and noise level are other examples of such time-varying parameters. These parameters are global in the sense that they are observations of the sound without any analytical separation of these components, which will be discussed in Chapter 10. They are related to perceptive cues and are based on hearing and psychoacoustics. These global parameters can be extracted from time-frequency or source-filter representations using classical tools of signal processing, where psychoacoustic fundamentals also have to be taken into account.

The use of these parameters for digital audio effects is twofold: one can use them inside the effect algorithm itself, or one can use these features as control variables for other effects. The latter technique will be described in Chapter 12. Pitch tracking as a source of control is a well-known application. Examples of audio effects using feature extraction inside the algorithm are the correction of tuning, which uses pitch extraction (autotune), or even the compression of a sound, which uses amplitude extraction.

9.4.1 Pitch Extraction

The main task of pitch extraction is to estimate a fundamental frequency f_0 , which in musical terms is the pitch of a sound segment, and follow the fundamental frequency over the time. We can use this pitch information to control effects like time stretching and pitch shifting based on the PSOLA method, which is described in Chapter 5 but it plays also a major role in sound modeling with spectral models, which is treated extensively in Chapter 10. Moreover, the fundamental frequency can be used as a control parameter for a variety of audio effects based either on time domain or on frequency domain processing.

There is no definitive technique for pitch extraction and tracking, and only the bases of existing algorithms will be described here. We will consider pitch extraction both in the frequency domain and in the time domain. Most often an algorithm first looks for candidates of a pitch, then selects one and tries to improve the precision of the choice. After the calculation of pitch candidates a post-processing, for example, pitch tracking has to be applied. During post-processing the estimation of the fundamental frequency from the pitch candidates can be improved by taking the frequency relationships between the detected candidates into account, which should ideally be multiples of the fundamental frequency.

FFT-based Approach

In this subsection we describe the calculation of pitch candidates from the FFT of a signal segment where the phase information is used. This approach is similar to the technique used in the phase vocoder, see section 8.3. The main structure of the algorithm is depicted in Fig. 9.27, where a segment of length N is extracted every R samples and then applied to FFTs.

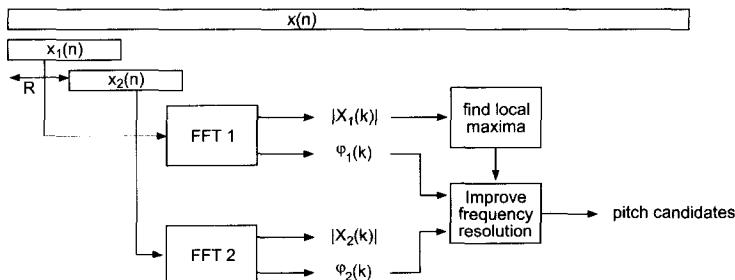


Figure 9.27 FFT-based pitch estimation structure with phase evaluation.

Considering the calculation of an N -point FFT, the frequency resolution of the FFT is

$$\Delta f = \frac{f_s}{N} \quad (9.30)$$

with the sampling frequency $f_s = 1/T_s$. From the input signal $x(n)$ we use a block

$$x_1(n) = x(n_0 + n) \quad , n = 0, \dots, N - 1 \quad (9.31)$$

of N samples. After applying an appropriate window, the FFT yields $X_1(k)$ with $k = 0, \dots, N - 1$. At the FFT index k_0 a local maximum of the FFT magnitude $|X_1(k)|$ is detected. From this FFT maximum, the initial estimate of the fundamental frequency is

$$\tilde{f}_0 = k_0 \cdot \Delta f = k_0 \frac{f_s}{N}. \quad (9.32)$$

The corresponding normalized frequency is

$$\tilde{\Omega}_0 = 2\pi \tilde{f}_0 T_s = k_0 \frac{2\pi}{N}. \quad (9.33)$$

To improve the frequency resolution, the phase information can be used, since for a harmonic signal $x_h(n) = \cos(\Omega_0 n + \varphi_0) = \cos(\phi(n))$ the fundamental frequency can be computed by the derivative

$$\Omega_0 = \frac{d\phi(n)}{dn}. \quad (9.34)$$

The derivative can be approximated by computing the phases of two FFTs separated by a hop size of R samples leading to

$$\hat{\Omega}_0 = \frac{\Delta\phi}{R}, \quad (9.35)$$

where $\Delta\phi$ is the phase difference between the two FFTs evaluated at the FFT index k_0 . The second FFT of the signal segment

$$x_2(n) = x(n_0 + R + n) \quad , n = 0, \dots, N - 1 \quad (9.36)$$

leads to $X_2(k)$. For the two FFTs, the phases at frequency \tilde{f}_0 are given by

$$\varphi_1 = \angle\{X_1(k_0)\} \quad (9.37)$$

$$\varphi_2 = \angle\{X_2(k_0)\}. \quad (9.38)$$

Both phases φ_1 and φ_2 are obtained in the range $[-\pi, \pi]$. We now calculate an “unwrapped” φ_2 value corresponding to the value of an instantaneous phase, see also section 8.3.5 and Fig. 8.17. Assuming that the signal contains a harmonic component with a frequency $\tilde{f}_0 = k_0 \cdot \Delta f$, the expected target phase after a hop size of R samples is

$$\varphi_{2t} = \varphi_1 + \tilde{\Omega}_0 R = \varphi_1 + \frac{2\pi}{N} k_0 R. \quad (9.39)$$

The phase error between the unwrapped value φ_2 and the target phase can be computed by

$$\varphi_{2err} = \text{princarg}(\varphi_2 - \varphi_{2t}). \quad (9.40)$$

The function “princarg” computes the principal phase argument in the range $[-\pi, \pi]$. It is assumed that the unwrapped phase differs from the target phase by a maximum of π . The unwrapped phase is obtained by

$$\varphi_{2u} = \varphi_{2t} + \varphi_{2err}. \quad (9.41)$$

The final estimate of the fundamental frequency is then obtained by

$$\hat{f}_0 = \frac{1}{2\pi} \hat{\Omega}_0 \cdot f_s = \frac{1}{2\pi} \cdot \frac{\varphi_{2u} - \varphi_1}{R} \cdot f_s. \quad (9.42)$$

Normally we assume that the first pitch estimation \tilde{f}_0 differs from the fundamental frequency by a maximum of $\Delta f/2$. Thus the maximum amount for the absolute value of the phase error φ_{2err} is

$$\varphi_{2err,max} = \frac{1}{2} \frac{2\pi}{N} R = \frac{R}{N} \pi. \quad (9.43)$$

We should accept phase errors with slightly higher values to have some tolerance in the pitch estimation.

One simple example of an ideal sine wave at a fundamental frequency of 420 Hz at $f_s = 44.1$ kHz analyzed with the FFT length $N = 1024$ using a Hanning window and a hop size $R = 1$ leads to the following results: $k_0 = 10$, $\tilde{f}_0 = k_0 \frac{f_s}{N} = 430.66$ Hz, $\varphi_1/\pi = -0.2474$, $\varphi_{2t}/\pi = -0.2278$, $\varphi_2/\pi = -0.2283$, $\hat{f}_0 = 419.9996$ Hz. Thus the original sine frequency is almost ideally recovered by the described algorithm.

Figure 9.28 shows an example of the described algorithm applied to a short signal of the female utterance “la” analyzed at an FFT length $N = 1024$. The top plot shows the FFT magnitude, the middle plot the estimated pitch, and the bottom plot the phase error φ_{2err} for frequencies up to 1500 Hz. For this example the frequency evaluation is performed for all FFT bins and not only for those with detected magnitude maxima. The circles show the positions of detected maxima in the FFT magnitude. The dashed lines in the bottom plot show the used threshold for the phase error. In this example the first maximum is detected at FFT index $k_0 = 6$, the corresponding bin frequency is 258.40 Hz, and the corrected pitch frequency is 274.99 Hz. Please notice that in this case the magnitude of the third harmonic (at appr. 820 Hz) has a greater value than the magnitude of the fundamental frequency.

M-file 9.15 presents a Matlab implementation to calculate the pitch candidates from a block of the input signal.

```
M-file 9.15 (find_pitch_fft.m)
function [FFTidx, Fp_est, Fp_corr]=...
find_pitch_fft(x, win, Nfft, Fs, R, fmin, fmax, thres)

%===== find pitch candidates =====
%      x: input signal of length Nfft+R
%      win: window for the FFT
```

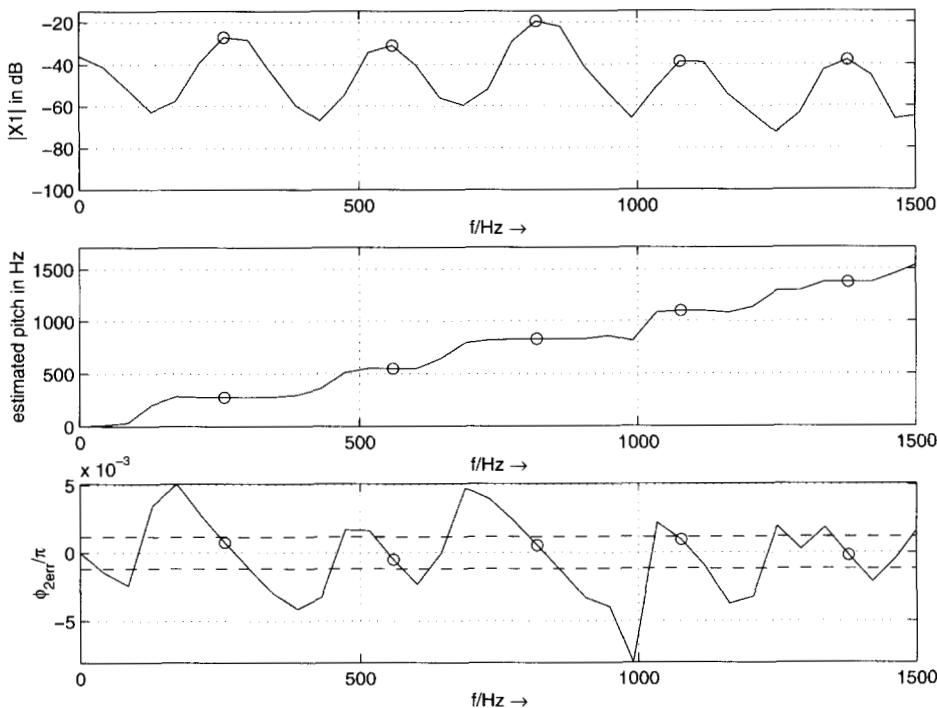


Figure 9.28 Example of pitch estimation of speech signal “la”.

```

%
% Nfft: FFT length
%
% Fs: sampling frequency
%
% R: FFT hop size
%
% fmin, fmax: minimum/ maximum pitch freqs to be detected
%
% thres: %omit maxima more than thres dB below the main peak

FFTidx = [];% FFT indices
Fp_est = [];% FFT bin frequencies
Fp_corr = [];% corrected frequencies
dt = R/Fs;% time diff between FFTs
df = Fs/Nfft;% freq resolution
kp_min = round(fmin/df);
kp_max = round(fmax/df);
x1 = x(1:Nfft);% 1st block
x2 = x((1:Nfft)+R);% 2nd block with hop size R
[X1, Phi1] = fftdb(x1.*win,Nfft);
[X2, Phi2] = fftdb(x2.*win,Nfft);
X1 = X1(1:kp_max+1);
Phi1 = Phi1(1:kp_max+1);
X2 = X2(1:kp_max+1);

```

```

Phi2      = Phi2(1:kp_max+1);
idx       = find_loc_max(X1);
Max       = max(X1(idx));
ii        = find(X1(idx)-Max>-thres);

%---- omit maxima more than thres dB below the main peak ----
idx       = idx(ii);
Nidx     = length(idx);    % number of detected maxima
maxerr   = R/Nfft;         % max phase diff error/pi
                           % (pitch max. 0.5 bins wrong)
maxerr   = maxerr*1.2;     % some tolerance
for ii=1:Nidx
    k           = idx(ii) - 1;          % FFT bin with maximum
    phi1        = Phi1(k+1);           % phase of x1 in [-pi,pi]
    phi2_t     = phi1 + 2*pi/Nfft*k*R; % expected target phase
                                       % after hop size R
    phi2        = Phi2(k+1);           % phase of x2 in [-pi,pi]
    phi2_err = princarg(phi2-phi2_t);
    phi2_unwrap = phi2_t+phi2_err;
    dphi      = phi2_unwrap - phi1;   % phase diff
    if (k>kp_min) & (abs(phi2_err)/pi<maxerr)
        Fp_corr = [Fp_corr; dphi/(2*pi*dt)];
        FFTIdx = [FFTIdx; k];
        Fp_est = [Fp_est; k*df];
    end
end

```

In addition to the algorithm described, the magnitude values of the detected FFT maxima are checked. In the given code those maxima are omitted whose FFT magnitudes are more than `thres` dB below the global maximum. Typical values for the parameter `thres` lie in the range from 30 to 50. The function `princarg` is given in Figure 8.17. The following function `fftdb` (see M-file 9.16) returns the FFT magnitude in a dB scale and the phase.

```

M-file 9.16 (fftdb.m)
function [H, phi] = fftdb(x, Nfft)

if nargin<2
    Nfft = length(x);
end

F      = fft(x,Nfft);
F      = F(1:Nfft/2+1);        % f=0,...,Fs/2
phi    = angle(F);            % phase in [-pi,pi]
F      = abs(F)/Nfft*2;       % normalize to FFT length
%---- return -100 db for F==0 to avoid "log of zero" warnings ----

```

```
H      = -100*ones(size(F));
idx   = find(F~=0);
H(idx) = 20*log10(F(idx));    % non-zero values in dB
```

The following function `find_loc_max` (see M-file 9.17) searches for local maxima using the derivative.

```
M-file 9.17 (find_loc_max.m)
function [idx, idx0] = find_loc_max(x)

% === find local maxima in vector x
%     idx : positions of local max.
%     idx0: positions of local max. with 2 identical values
%     if only 1 return value: positions of all maxima

N      = length(x);
dx    = diff(x);           % derivation
                           % to find sign changes from + to -
dx1   = dx(2:N-1);
dx2   = dx(1:N-2);
prod  = dx1.*dx2;
idx1  = find(prod<0);    % sign change in dx1
idx2  = find(dx1(idx1)<0); % only change from + to -
idx   = idx1(idx2)+1;      % positions of single maxima
%---- zeros in dx? => maxima with 2 identical values ----
idx3  = find(dx==0);
idx4  = find(x(idx3)>0); % only maxima
idx0  = idx3(idx4);
%---- positions of double maxima, same values at idx3(idx4)+1 ----
if nargout==1             % output 1 vector
                           % with positions of all maxima
    idx = sort([idx, idx0]); % (for double max. only 1st position)
end
```

Now we present an example where the algorithm is applied to a signal segment of Suzanne Vega’s “Tom’s Diner”. Figure 9.29 shows time-frequency representations of the analysis results. The top plot shows the spectrogram of the signal. The middle plot shows the FFT bin frequencies of detected pitch candidates while the bottom plot shows the corrected frequency values. In the bottom plot the text of the sung words is also shown. In all plots frequencies up to 800 Hz are shown. For the spectrogram an FFT length of 4096 points is used. The pitch estimation algorithm is performed with an FFT length of 1024 points. This example shows that the melody of the sound can be recognized in the bottom plot of Figure 9.29. The applied algorithm improves the frequency resolution of the FFT shown in the middle plot. To choose the correct pitch among the detected candidates some post-processing is required. Other methods to improve the frequency resolution of the FFT are described in [Can98, DM00, Mar00, Mar98, AKZ99] and in Chapter 10.

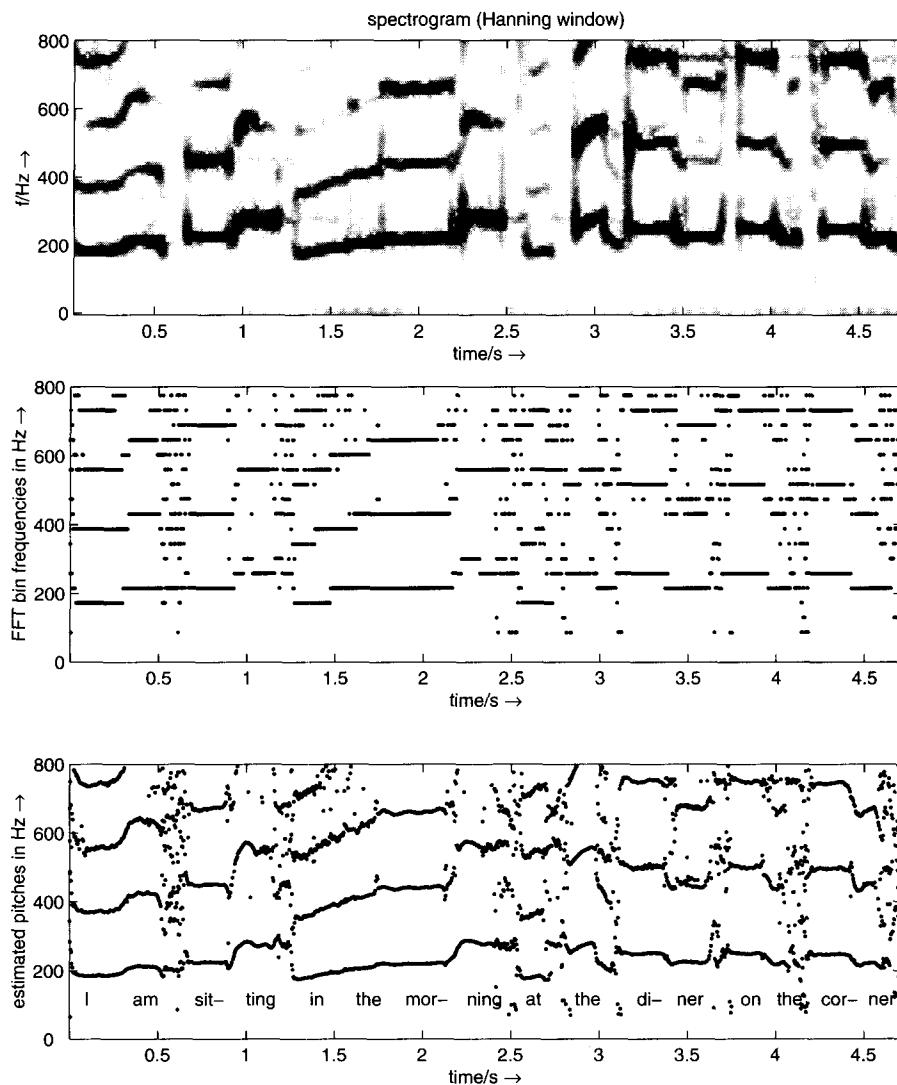


Figure 9.29 Time/frequency planes for pitch estimation example of an excerpt from Suzanne Vega's "Tom's Diner". Top: spectrogram, middle: FFT bin frequencies of pitch candidates, bottom: corrected frequency values of pitch candidates.

M-file 9.18 demonstrates a pitch tracking algorithm in a block-based implementation.

M-file 9.18 (Pitch_Tracker_FFT_Main.m)

```
% Pitch_Tracker_FFT_Main.m
fname='Toms_diner';
```

```

n0=2000; %start index
n1=210000;

Nfft=1024;
R=1;          % FFT hop size for pitch estimation
K=200;        % hop size for time resolution of pitch estimation
thres=50; % threshold for FFT maxima
% checked pitch range in Hz:
fmin=50;
fmax=800;
p_fac_thres=1.05; % threshold for voiced detection
                  % deviation of pitch from mean value
win=hanning(Nfft)';% window for FFT
Nx=n1-n0+1+R;      % signal length
blocks=floor(Nx/K);
Nx=(blocks-1)*K+Nfft+R;
n1=n0+Nx;           % new end index
[X,Fs]=wavread(fname,[n0,n1]);
X=X(:,1)';

pitches=zeros(1,blocks);
for b=1:blocks
    x=X((b-1)*K+1+(1:Nfft+R));
    [FFTidx, F0_est, F0_corr]= ...
        find_pitch_fft(x,win,Nfft,Fs,R,fmin,fmax,thres);
    if ~isempty(F0_corr)
        pitches(b)=F0_corr(1); % take candidate with lowest pitch
    else
        pitches(b)=0;
    end
end
%%%% post-processing:
L=9;          % odd number of blocks for mean calculation
D=(L-1)/2;    % delay
h=ones(1,L)./L; % impulse response for mean calculation
% mirror beginning and end for "non-causal" filtering:
p=[pitches(D+1:-1:2),pitches,pitches(blocks-1:-1:blocks-D)];
y=conv(p,h);      % length: blocks+2D+2D
pm=y((1:blocks)+2*D); % cut result

Fac=zeros(1,blocks);
idx=find(pm~=0); % don't divide by zero
Fac(idx)=pitches(idx)./pm(idx);
ii=find(Fac<1 & Fac~=0);
Fac(ii)=1./Fac(ii); % all non-zero elements are now > 1
% voiced/unvoiced detection:

```

```

voiced=Fac~=0 & Fac<p_fac_thres;

T=40; % time in ms for segment lengths
M=round(T/1000*Fs/K); % min. number of consecutive blocks
[V,p2]=segmentation(voiced, M, pitches);
p2=V.*p2; % set pitches to zero for unvoiced

figure(1),clf,
time=(0:blocks-1)*K+1; % start sample of blocks
time=time/Fs; % time in seconds
t=(0:length(X)-1)/Fs; % time in sec for original
subplot(211)
plot(t,X),title('original x(n)')
axis([0 max([t,time]) -1.1*max(abs(X)) 1.1*max(abs(X))])
subplot(212)
idx=find(p2~=0);
plot_split(idx,time, p2),title('pitch in Hz');
xlabel('time/s \rightarrow');
axis([0 max([t,time]) .9*min(p2(idx)) 1.1*max(p2(idx))])

```

In the above implementation the post-processing is performed by choosing the lowest pitch candidate in each block. Then the mean pitch of surrounding blocks is computed and compared to the detected pitch. If the deviation from the mean value is higher than a given threshold, this block is considered as “unvoiced”. Finally a segmentation is performed to get a minimum number of consecutive blocks that are voiced/unvoiced (to avoid very short segments). M-file 9.19 presents an implementation for the segmentation.

M-file 9.19 (segmentation.m)

```

function [V,pitches2]=segmentation(voiced, M, pitches)
% voiced: original voiced/unvoiced detection
% M: min. number of consecutive blocks with same voiced flag
% pitches: original pitches
% V: changed voiced flag
% pitches2: changed pitches

blocks=length(voiced); % get number of blocks
pitches2=pitches;
V=voiced;
Nv=length(V);

%%%%%% step1: eliminate too short voiced segments:
V(Nv+1)=~V(Nv); % change at end to get length of last segment
dv=[0, diff(V)]; % derivative
idx=find(dv~=0); % changes in voiced
di=[idx(1)-1,diff(idx)]; % segment lengths

```

```

v0=V(1);           % status of 1st segment
k0=1;
ii=1; % counter for segments, idx(ii)-1 is end of segment
if v0==0
    k0=idx(1); % start of voiced
    ii=ii+1;   % first change voiced to unvoiced
end
while ii<=length(idx);
    L=di(ii);
    k1=idx(ii)-1; % end of voiced segment
    if L<M
        V(k0:k1)=zeros(1,k1-k0+1);
    end
    if ii<length(idx)
        k0=idx(ii+1); % start of next voiced segment
    end
    ii=ii+2;
end

%%%%%%% step2: eliminate too short unvoiced segments:
V(Nv+1)=~V(Nv);           % one more change at end
dv=[0, diff(V)];
idx=find(dv~=0);           % changes in voiced
di=[idx(1)-1,diff(idx)];  % segment lengths
if length(idx)>1           % changes in V
    v0=V(1);               % status of 1st segment
    k0=1;
    ii=1; % counter for segments, idx(ii)-1 is end of segment
    if v0==0
        k0=idx(2); % start of unvoiced
        ii=ii+2;   % first change unvoiced to voiced
    end
    while ii<=length(idx);
        L=di(ii);
        k1=idx(ii)-1; % end of unvoiced segment
        if L<M
            if k1<blocks % NOT last unvoiced segment
                V(k0:k1)=ones(1,k1-k0+1);
                % linear pitch interpolation:
                p0=pitches(k0-1);
                p1=pitches(k1+1);
                N=k1-k0+1;
                pitches2(k0:k1)=(1:N)*(p1-p0)/(N+1)+p0;
            end
        end
        if ii<length(idx)

```

```

k0=idx(ii+1); % start of next unvoiced segment
end
ii=ii+2;
end
end

V=V(1:Nv); % cut last element

```

The plot_split function is given by M-file 9.20.

```

M-file 9.20 (plot_split.m)
function plot_split(idx, t, x)
% idx: vector with positions of vector x to be plotted
% x is segmented into parts
di=diff(idx);
L=length(di);

n0=1;
pos_di=find(di>1);
ii=1; % counter for pos_di

hold off
while ii<=length(pos_di) %n0<=length(x)
    n1=pos_di(ii);
    plot(t(idx(n0:n1)),x(idx(n0:n1)))
    hold on
    n0=n1+1;
    ii=ii+1;
end

n1=length(idx);
plot(t(idx(n0:n1)),x(idx(n0:n1)))
hold off

```

The result of the pitch tracking algorithm is illustrated in Fig. 9.30. The bottom plot shows the pitch over time calculated using the block-based FFT approach.

Any FFT-based pitch estimator can be improved by detecting the harmonic structure of the sound. If the harmonics of the fundamental frequency are detected, the greatest common divisor of these harmonic frequencies can be used in the estimation of the fundamental frequency [O'S00, p. 220]. M.R. Schroeder mentions for speech processing in [Sch99, p. 65], “the pitch problem was finally laid to rest with the invention of cepstrum pitch detectors” [Nol64]. The cepstrum technique allows the estimation of the pitch period directly from the cepstrum sequence $c(n)$. Schroeder also suggested a “harmonic product spectrum” [Sch68] to improve the fundamental frequency estimation, which sometimes outperforms the cepstrum method [Sch99, p. 65]. A further improvement of the pitch estimates can

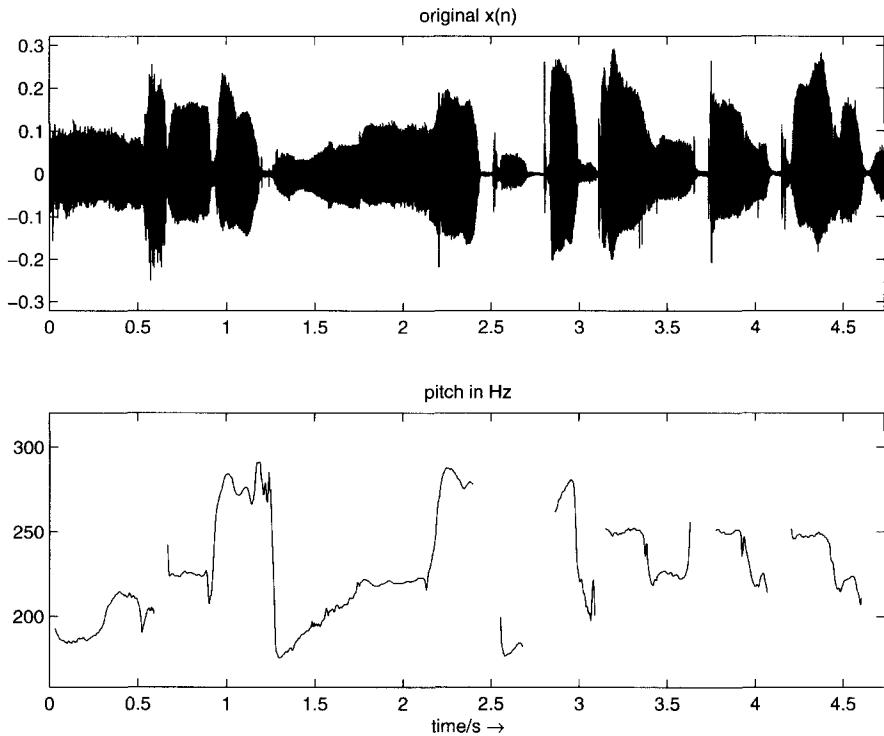


Figure 9.30 Pitch over time from the FFT with phase vocoder approach for a signal segment of Suzanne Vega’s “Tom’s Diner”.

be achieved by applying a peak continuation algorithm to the detected pitches of adjacent frames, which is described in section 10.3.1.

General Remarks on Time-Domain Pitch Extraction

In the time domain the task of pitch extraction leads us to find the corresponding *pitch period*. The pitch period is the time duration of one period. With the fundamental frequency f_0 (to be detected) the pitch period is given by

$$T_0 = \frac{1}{f_0}. \quad (9.44)$$

For a discrete-time signal sampled at $f_s = \frac{1}{T_s}$ we have to find the *pitch lag* M which is the number of samples in one period. The pitch period is $T_0 = M \cdot T_s$ which leads to

$$M = \frac{T_0}{T_s} = \frac{f_s}{f_0}. \quad (9.45)$$

Since only integer-valued pitch lags can be detected, we have a certain frequency resolution in detecting the fundamental frequency dependent on f_0 and f_s . Now we are assuming the case of $\tilde{M} = M + 0.5$ where \tilde{M} is the detected integer pitch lag. The detected fundamental frequency is $\tilde{f}_0 = \frac{f_s}{\tilde{M}}$ instead of the exact pitch $f_0 = \frac{f_s}{M}$. The frequency error factor is in this case

$$\alpha(f_0) = \frac{f_0}{\tilde{f}_0} = \frac{\frac{f_s}{M}}{\frac{f_s}{\tilde{M}}} = 1 + \frac{0.5}{M} = 1 + 0.5 \frac{f_0}{f_s}. \quad (9.46)$$

With the halftone factor $\alpha_{ht} = \sqrt[12]{2}$ and setting $\alpha(f_0) = \alpha_{ht}^x$, the frequency error in halftones is

$$x = \frac{\ln \alpha(f_0)}{\ln \alpha_{ht}}. \quad (9.47)$$

Figure 9.31 shows the frequency error both as factor $\alpha(f_0)$ and as percentage of halftones for pitches in the range from 50 to 5000 Hz at the sampling frequency $f_s = 44.1$ kHz. The maximum frequency error is approximately 6 percent or one halftone for pitches up to 5000 Hz. For a fundamental frequency of 1000 Hz the frequency error is only 20 percent of a halftone which is reasonably accurate precision.

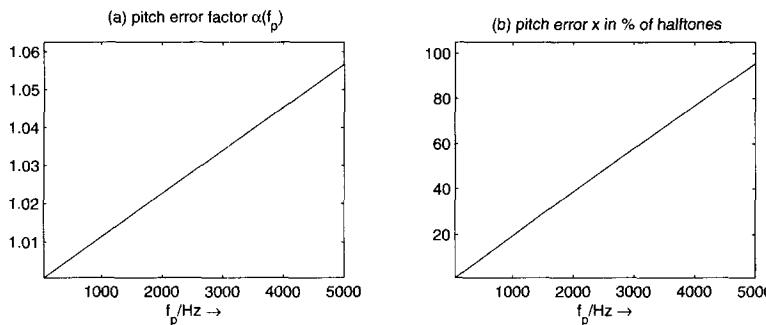


Figure 9.31 Resolution of time-domain pitch detection at $f_s = 44.1$ kHz, (a) frequency error factor, (b) pitch error in percentage of a halftone.

Normally the pitch estimation in the time domain is performed in three steps [O'S00, p. 218]:

1. Segmentation of the input signal into overlapping blocks and pre-processing of each block, for example lowpass filtering (see segmentation shown in Fig. 9.27).
2. Basic pitch estimation algorithm applied to the pre-processed block.
3. Post-processing for an error correction of the pitch estimates and smoothing of pitch trajectories.

Autocorrelation and LPC

The autocorrelation sequence can also be used to detect the pitch period of a signal segment. First, we present different definitions of autocorrelation sequences:

- Using one block

$$r_{xx}(m) = \sum_{n=m}^{N-1} x(n)x(n-m), \quad (9.48)$$

- using one windowed block

$$r_{xx}(m) = \sum_{n=m}^{N-1} u(n)u(n-m) \quad (9.49)$$

with $u(n) = x(n) \cdot w(n)$ (window function $w(n)$)

- and using the exact signal, thus using samples preceding the considered block

$$\tilde{r}_{xx}(m) = \sum_{n=0}^{N-1} x(n)x(n-m). \quad (9.50)$$

Notice, that in the definitions given by (9.48)–(9.50) no normalization to the block length N is applied.

Figure 9.32 shows the three different autocorrelation sequences for an excerpt of the speech signal “la”. Here the same input signal is used as in Fig. 9.28. In this example the pitch lag corresponding to the fundamental frequency is $M = 160$ samples, thus at the third maximum of the autocorrelation. Normally we expect the first maximum in the autocorrelation at the pitch lag. But sometimes, as in this example, the first maximum in the autocorrelation function is not at this position. In general, the autocorrelation has maxima at the pitch lag M and at its multiples since, for a periodic signal, the same correlation occurs if comparing the signal with the same signal delayed by multiples of the pitch period. Since, in the example of Figures 9.28 and 9.32, the third harmonic is more dominant than the fundamental frequency, the first maximum in the autocorrelation is located at $M/3$. Conversely there can be a higher peak in the autocorrelation after the true pitch period.

Often the prediction error of an LPC analysis contains peaks spaced by the pitch period, see Fig. 9.9. Thus it might be promising to try to estimate the pitch period from the prediction error instead of using the original signal. The SIFT algorithm [Mar72], which has been developed for voice, is based on removing the spectral envelope by inverse filtering in a linear prediction scheme. But in some cases it is not possible to estimate the pitch period from the prediction error, because the linear prediction has removed all pitch redundancies from the signal. Figure 9.33 compares two excerpts of a speech signal where the input block (top) and the autocorrelations of both the input signal (middle) and the prediction error (bottom) are shown. An LPC analysis of order $p = 8$ using the autocorrelation method has been applied.

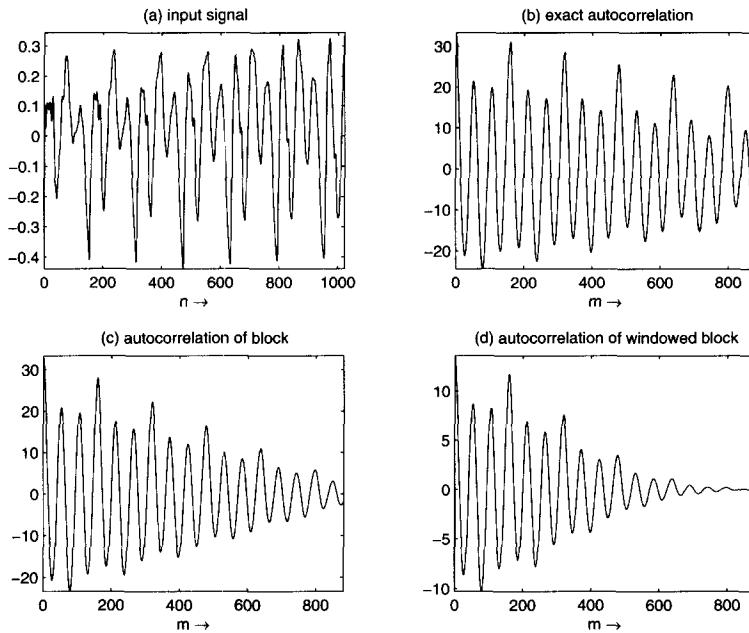


Figure 9.32 Comparison between different autocorrelation computations for speech signal “la”. (a) input block $x(n)$, (b) exact autocorrelation $\tilde{r}_{xx}(m)$, (c) standard autocorrelation $r_{xx}(m)$ using block, (d) standard autocorrelation $r_{xx}(m)$ using windowed block.

For the example presented in subplots (a)–(c) the pitch period can be well detected in the autocorrelation of the prediction error (same excerpt as in Figures 9.28 and 9.32). For the other excerpt presented in subplots (d)–(f) it is not possible to detect the pitch in the autocorrelation of the prediction error while the autocorrelation of the input signal has a local maximum at the correct pitch period. Notice that in the plots of the autocorrelation sequences only time lag ranges from 29 to 882 are shown. This corresponds to pitch frequencies from 1500 down to 50 Hz at sampling frequency 44.1 kHz.

Another time-domain method for the extraction of the fundamental frequency is based on “center clipping” the input signal and subsequent autocorrelation analysis [Son68]. First, the input signal is bandlimited by a lowpass filter. If the filter output signal exceeds a certain threshold $\pm c$ the operation $x_{clip}(n) = x(n) \mp c$ is performed, otherwise $x_{clip}(n) = 0$ [RS78, Son68]. The result of this pre-processing is illustrated in Fig. 9.34. The autocorrelation sequence $r_{xx}(m)$ of the center clipped signal $x_{clip}(n)$ shows a strong positive peak at the time lag of the pitch period.

Long-Term Prediction (LTP)

A further method of estimating the fundamental frequency is based on long-term prediction. A common approach to remove pitch period redundancies from a signal

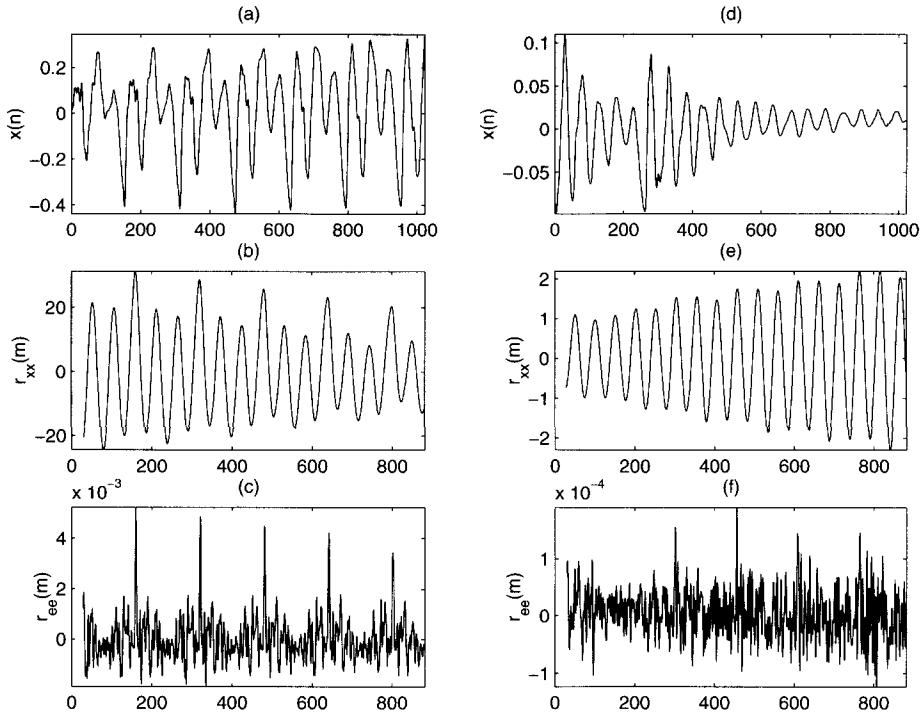


Figure 9.33 Autocorrelation sequences for input signal and prediction error for two excerpts of the speech signal “la”. Input signals (a, d), autocorrelation of input (b, e), autocorrelation of prediction error (c, f).

is to use a short FIR prediction filter after a delay line of M samples, where M is the pitch lag [KA90]. Thus the long-term prediction error or residual is given by

$$d(n) = x(n) - \sum_{k=0}^q b_k x(n-M-k), \quad (9.51)$$

where the order $q + 1$ is normally in the range of $\{1, 2, 3\}$ [KA90]. Considering the case of a one-tap filter, the residual simplifies to

$$d(n) = x(n) - b_0 \cdot x(n-M) \quad (9.52)$$

which is shown in Fig. 9.35.

For minimizing the energy of $d(n)$ over one block of length N we set the derivative with respect to b_0 to zero. This leads to the optimal filter coefficient

$$b_0 = \frac{\hat{r}_{xx}(M)}{r_{xx0}(M)} \quad (9.53)$$

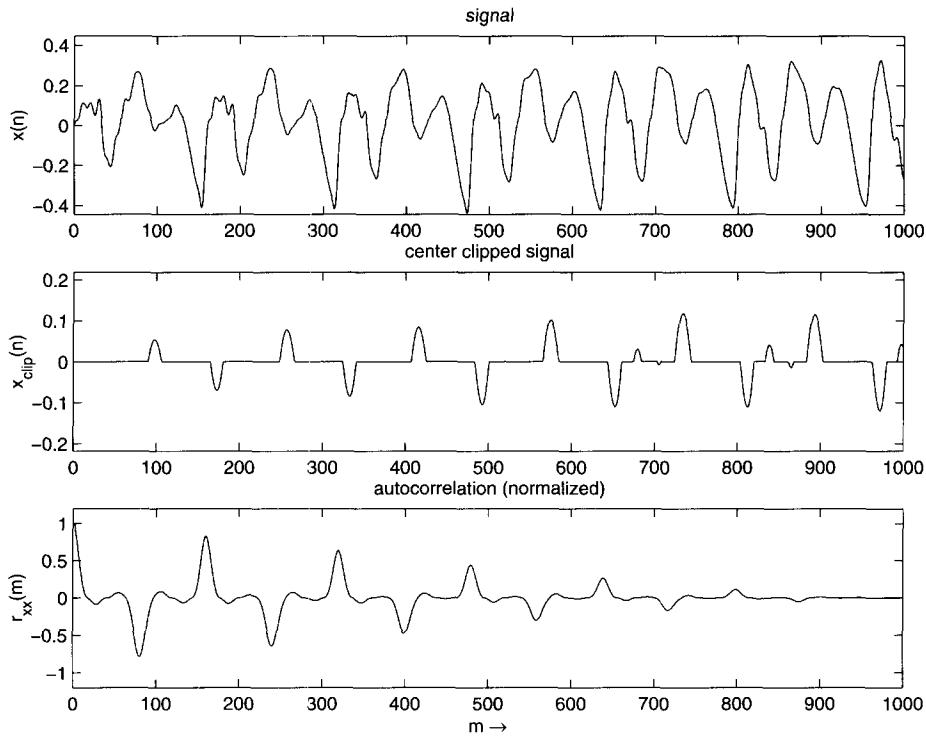


Figure 9.34 Center clipping and subsequent autocorrelation analysis: input signal, low-pass filtered and center clipped signal (notice the time delay) and autocorrelation.

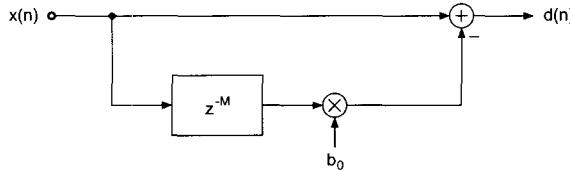


Figure 9.35 Long-term prediction with a one-tap filter.

with $\tilde{r}_{xx}(m)$ as defined in (9.50) and

$$r_{xx0}(m) = \sum_{n=0}^{N-1} x^2(n - m) \quad (9.54)$$

which is the energy of a block delayed by m samples. Setting this solution into (9.52) leads to the error energy

$$E_d = \sum_{n=0}^{N-1} x^2(n) - r_{xx,norm}(M) \quad (9.55)$$

dependent on M with

$$r_{xx,norm}(m) = \frac{\tilde{r}_{xx}^2(m)}{r_{xx0}(m)}. \quad (9.56)$$

Figure 9.36 shows an example where the input signal shown in Figure 9.33(a) is used. The top plot shows the exact autocorrelation $\tilde{r}_{xx}(m)$, the middle plot shows the normalized autocorrelation $r_{xx,norm}(m)$, and the bottom plot shows the LTP coefficient $b_0(m)$ dependent on the lag m .

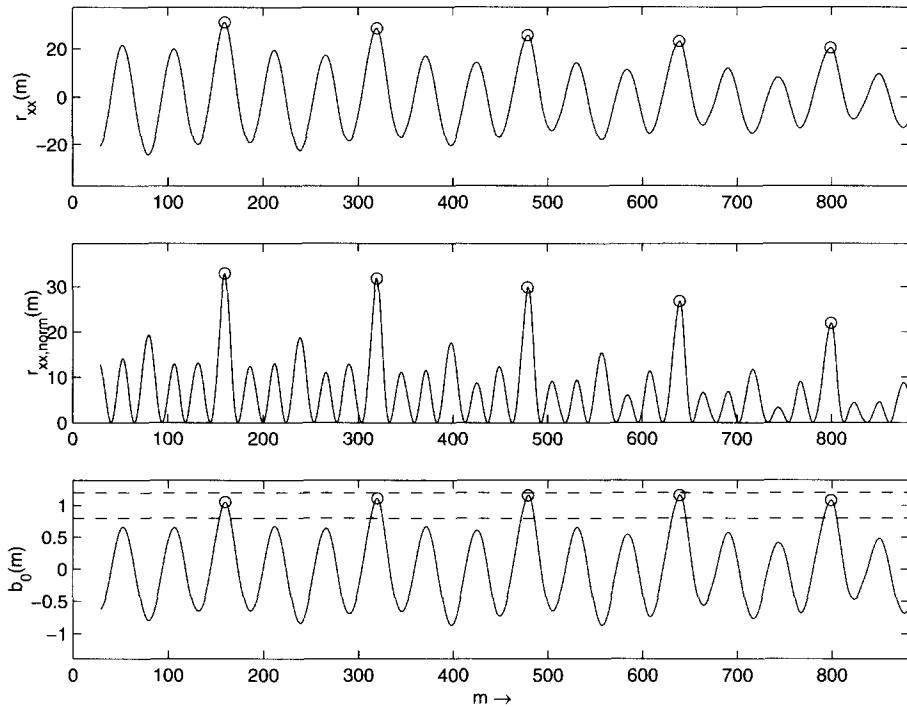


Figure 9.36 Autocorrelation, normalized autocorrelation and LTP coefficient dependent on lag m for excerpt from the speech signal “la”. The circles show the pitch lag candidates, the dashed lines the accepted b_0 values.

In $r_{xx,norm}(m)$ the lag $m = M$ has to be found where $r_{xx,norm}(m)$ is maximized to minimize the residual energy. Considering one block of length N , the numerator of (9.56) is the squared autocorrelation while the denominator is the energy of the block delayed by M samples. The function $r_{xx,norm}(m)$ therefore represents a kind of normalized autocorrelation sequence with only positive values. If used for the detection of the pitch period, $r_{xx,norm}(m)$ does not need to have a global maximum at $m = M$, but it is expected to have a local maximum at that position.

To find candidates of the pitch lag M , first local maxima in $r_{xx,norm}(m)$ are searched. In a second step, from these maxima only those ones are considered where

the autocorrelation $\tilde{r}_{xx}(m)$ is positive valued. The function $r_{xx,norm}(m)$ also has maxima at positions where $\tilde{r}_{xx}(m)$ has minima. In a third step the $b_0(m)$ values are considered. The value of the coefficient b_0 is close to one for voiced sounds and close to zero for noise-like sounds [JN84, p. 315]. Thus the value of b_0 can serve as a quality check for the estimate of the computed pitch lag.

In the example in Figure 9.36, b_0 values in the range $0.8, \dots, 1.2$ are accepted. This range is shown by the dashed lines in the bottom plot. The circles represent the positions of pitch lag candidates. Thus, at these positions $r_{xx,norm}(m)$ has a local maximum, $\tilde{r}_{xx}(m)$ is positive valued, and $b_0(m)$ lies in the described range. In this example, the first pitch lag candidate corresponds to the pitch of the sound segment.

The described algorithm for the computation of pitch lag candidates from a signal block is implemented by the following M-file 9.21.

```
M-file 9.21 (find_pitch_ltp.m)
function [M,Fp]=find_pitch_ltp(xp,lmin,lmax,Nblock,Fs,b0_thres)

%      xp      : input block including lmax pre-samples
%                  for correct autocorrelation
%      lmin    : min. checked pitch lag
%      lmax    : max. checked pitch lag
%      Nblock  : block length without pre-samples
%      Fs      : sampling freq.
%      b0_thres: max b0 deviation from 1

lags = lmin:lmax;          % tested lag range
Nlag = length(lags);       % no. of lags
[rxx_norm, rxx, rxx0] = xcorr_norm(xp, lmin, lmax, Nblock);

%---- calc. autocorr sequences -----
B0 = rxx./rxx0;           % LTP coeffs for all lags
idx = find_loc_max(rxx_norm);
i = find(rxx(idx)>0); % only max. where r_xx>0
idx = idx(i);            % indices of maxima candidates
i = find(abs(B0(idx)-1)<b0_thres);

%---- only max. where LTP coeff is close to 1 -----
idx = idx(i);            % indices of maxima candidates

%---- vectors for all pitch candidates: -----
M = lags(idx);
M = M(:);                 % pitch lags
Fp = Fs./M;
Fp = Fp(:);               % pitch freqs
```

The function `find_loc_max` is given in section 9.4.1. The function `xcorr_norm`

to compute the autocorrelation sequences is given by M-file 9.22.

M-file 9.22 (xcorr_norm.m)

```
function [rxx_norm, rxx, rxx0] = xcorr_norm(xp, lmin, lmax, Nblock)
%==== calc. normalized autocorrelation=====

x      = xp((1:Nblock)+lmax); % input block without pre-samples
lags   = lmin:lmax;           % tested lag range
Nlag   = length(lags);        % no. of lags
rxx    = zeros(1,Nlag);       % autocorr. sequence
rxx0   = zeros(1,Nlag);       % energy of delayed blocks
rxx_norm = zeros(1,Nlag);     % normalized autocorr. sequence
for l=1:Nlag
    ii   = lags(l);           % tested lag
    rxx0(l) = sum(xp((1:Nblock)+lmax-lags(l)).^2);
    %---- energy of delayed block
    rxx(l) = sum(x.*xp((1:Nblock)+lmax-lags(l)));
end
rxx_norm=rxx.^2./rxx0;         % normalized autocorr. sequence
```

The performance of the function `xcorr_norm` is quite slow in Matlab. The computation speed can be improved if using a C-MEX function. Thus the function is implemented in C and a “MEX” file is created with the C compiler (on Windows systems the MEX file is a dll). In this example the computation speed is improved by a factor of approximately 50, if using the MEX file instead of the Matlab function.

The described LTP algorithm may also be applied to the prediction error of a linear prediction approach. Figure 9.37 compares LTP applied to original signals and their prediction errors. In this example the same signal segments are used as in Fig. 9.33. The circles denote the detected global maxima in the normalized autocorrelation. For the first signal shown in plots (a)–(c) the computed LTP coefficients are $b_{0x} = 1.055$ for the input signal and $b_{0e} = 0.663$ for the prediction error. The LTP coefficients for the second signal are $b_{0x} = 0.704$ and $b_{0e} = 0.141$, respectively. As in Figure 9.33, the pitch estimation from the prediction error works well for the first signal while this approach fails for the second signal. For the second signal the value of the LTP coefficient indicates that the prediction error is noise-like.

Figure 9.38 shows the detected pitch lag candidates and the corresponding frequencies over time for a signal segment of Suzanne Vega’s “Tom’s Diner”. It is the same example as presented in Fig. 9.29 where also a spectrogram of this sound signal is given. The top plot of Fig. 9.38 shows the detected pitch lag candidates computed by the LTP algorithm applied to the input signal. The parameter `b0_thres` is set to 0.3, thus b_0 values in the range $0.7, \dots, 1.3$ are accepted. The corresponding pitch frequencies in the bottom plot are computed by $f_p = f_s/M$ (see (9.45)).

In the top plot of Figure 9.38 the lowest detected pitch lag normally corresponds to the pitch of the signal frame. The algorithm detects other candidates at multiples of this pitch lag. In some parts of this signal (for example, between 3

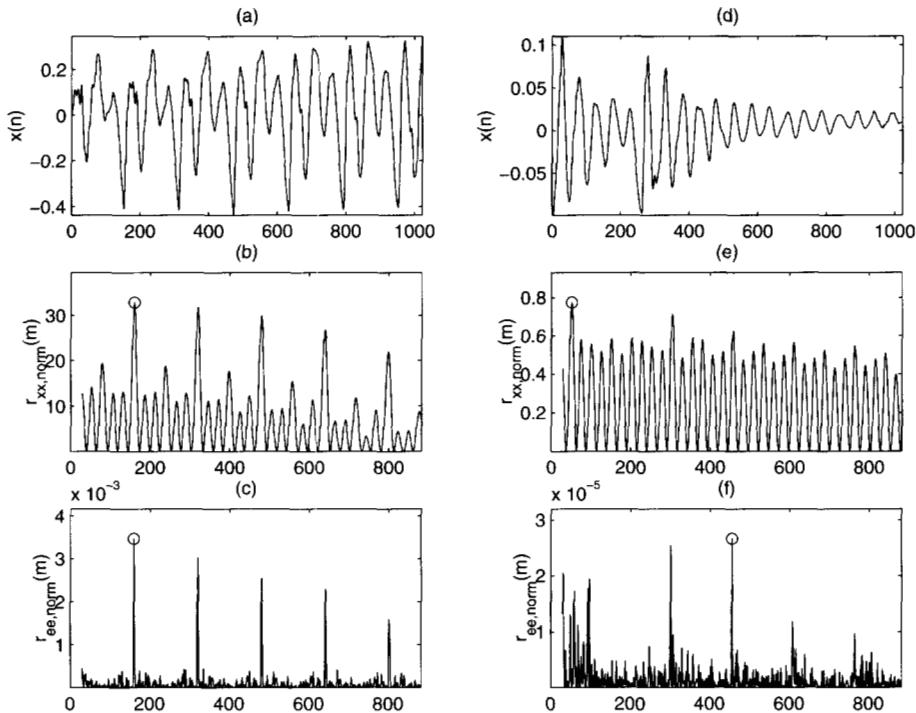


Figure 9.37 Normalized autocorrelation sequences for input signal and prediction error for two excerpts of the speech signal “la”. Input signals (a, d), normalized autocorrelation of input (b, e), normalized autocorrelation of prediction error (c, f).

and 4 seconds) the third harmonic of the real pitch is more dominant than the fundamental frequency. In these parts the lowest detected pitch lag is not the one to be chosen. In this time-domain approach the precision of the detected pitch lags can be improved if the greatest common divisor of the pitch lag candidates is used. The algorithm computes only integer-valued pitch lag candidates. LTP with a higher precision (noninteger pitch lag M) is presented in [LVKL96, KA90]. As in the FFT-based approach a post-processing should be applied to choose one of the detected candidates for each frame. For a more reliable pitch estimation both time and frequency domain approaches may be combined.

The following M-file 9.23 presents an implementation of a pitch tracker based on the LTP approach.

M-file 9.23 (Pitch_Tracker_LTP.m)

```

fname='Toms_diner';
n0=2000; % start index
n1=210000;
K=200;    % hop size for time resolution of pitch estimation
N=1024;   % block length

```

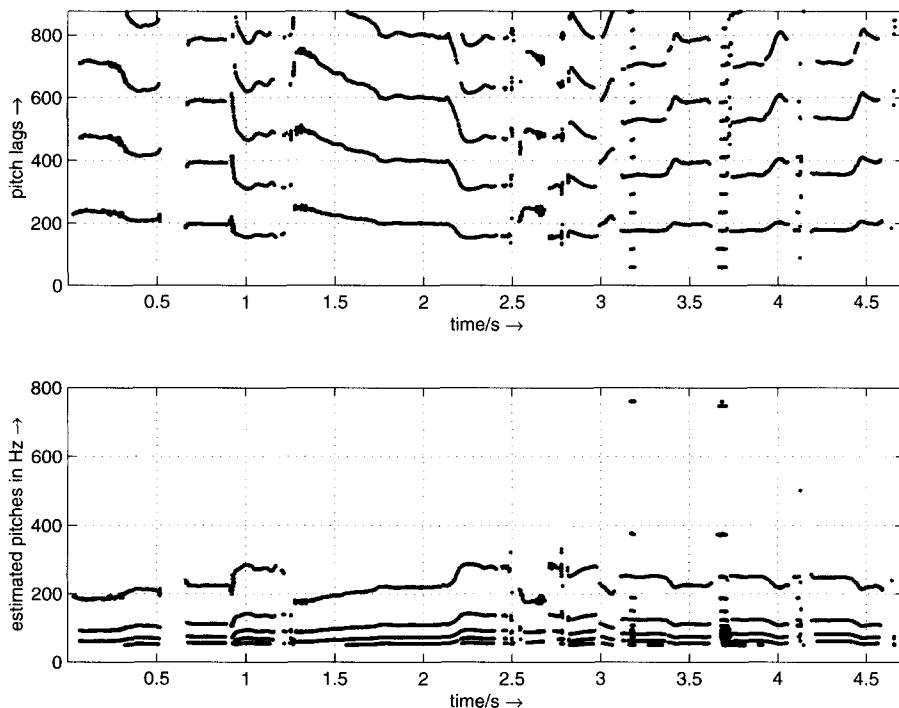


Figure 9.38 Pitch lag candidates and corresponding frequencies.

```
% checked pitch range in Hz:  
fmin=50;  
fmax=800;  
b0_thres=.2; % threshold for LTP coeff  
p_fac_thres=1.05; % threshold for voiced detection  
% deviation of pitch from mean value  
  
[xin,Fs]=wavread(fname,[n0 n0]); %get Fs  
% lag range in samples:  
lmin=floor(Fs/fmax);  
lmax=ceil(Fs/fmin);  
pre=lmax; % number of pre-samples  
if n0-pre<1  
    n0=pre+1;  
end  
Nx=n1-n0+1; % signal length  
blocks=floor(Nx/K);  
Nx=(blocks-1)*K+N;  
[X,Fs]=wavread(fname,[n0-pre n0+Nx]);
```

```
X=X(:,1)';

pitches=zeros(1,blocks);
for b=1:blocks
    x=X((b-1)*K+(1:N+pre));
    [M, F0]=find_pitch_ltp(x, lmin, lmax, N, Fs, b0_thres);
    if ~isempty(M)
        pitches(b)=Fs/M(1); % take candidate with lowest pitch
    else
        pitches(b)=0;
    end
end

%%%% post-processing:
L=9; % number of blocks for mean calculation
if mod(L,2)==0 % L is even
    L=L+1;
end
D=(L-1)/2; % delay
h=ones(1,L)./L; % impulse response for mean calculation
% mirror start and end for "non-causal" filtering:
p=[pitches(D+1:-1:2), pitches, pitches(blocks-1:-1:blocks-D)];
y=conv(p,h); % length: blocks+2D+2D
pm=y((1:blocks)+2*D); % cut result

Fac=zeros(1,blocks);
idx=find(pm~=0); % don't divide by zero
Fac(idx)=pitches(idx)./pm(idx);
ii=find(Fac<1 & Fac~=0);
Fac(ii)=1./Fac(ii); % all non-zero element are now > 1
% voiced/unvoiced detection:
voiced=Fac~=0 & Fac<p_fac_thres;

T=40; % time in ms for segment lengths
M=round(T/1000*Fs/K); % min. number of blocks in a row
[V,p2]=segmentation(voiced, M, pitches);
p2=V.*p2; % set pitches to zero for unvoiced

figure(1),clf;
time=(0:blocks-1)*K+1; % start sample of blocks
time=time/Fs; % time in seconds
t=(0:length(X)-1)/Fs; % time in sec for original
subplot(211)
plot(t, X),title('original x(n)');
axis([0 max([t,time]) -1.1*max(abs(X)) 1.1*max(abs(X))])
```

```

subplot(212)
idx=find(p2~=0);
plot_split(idx,time, p2),title('pitch in Hz');
xlabel('time/s \rightarrow');
axis([0 max([t,time]) .9*min(p2(idx)) 1.1*max(p2(idx))])

```

The result of the presented pitch tracking algorithm is illustrated in Fig. 9.39. The bottom plot shows the pitch over time calculated using the LTP method. In comparison to the FFT-based approach in Fig. 9.30, the FFT approach performs better in the regions where unvoiced parts occur. The described approach performs well for singing voice examples. The selection of the post-processing strategy depends on the specific sound or signal.

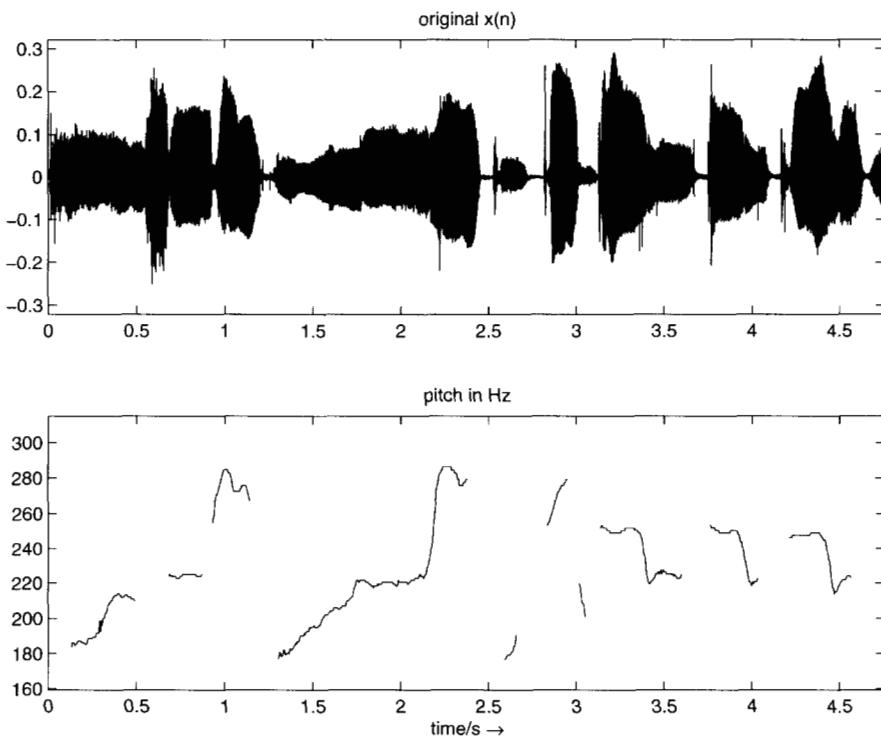


Figure 9.39 Pitch over time using the long-term prediction method for a signal segment of Suzanne Vega's "Tom's Diner".

9.4.2 Other Features

Amplitude Envelope

One very important feature that can be used for adaptive effects is the amplitude envelope of a sound evolving with time. Even the modulation of a sound by the envelope of another sound is an effect by itself. But more generally the amplitude envelope can be used to control many variables of an effect. Applications of amplitude detection can be found in dynamics processing (see Chapter 5) but can also be integrated into many effects as an external control parameter.

Except for the fact that we want to write a signal as $x(n) = \text{amp}(n) \cdot \text{sig}(n)$, there is no unique definition of an amplitude envelope of a sound. The ear is devised in such a way that slow variations of amplitude (under 10 Hz) are considered as a time envelope while more rapid variations would be heard as a sound. This distinction between an envelope and a signal is known in electroacoustic music as the difference between a “shape” and a “matter”, two terms well developed by P. Schaeffer in his *Traité des objets musicaux* [Sch66].

The RMS (root mean square) algorithm has been largely used in Chapter 5 as an amplitude detector based on filtering the squared input samples and taking the square root of the filter output. The RMS value is a good indication of the temporal variation of the energy of a sound, as shown in Fig. 9.40. This filtering

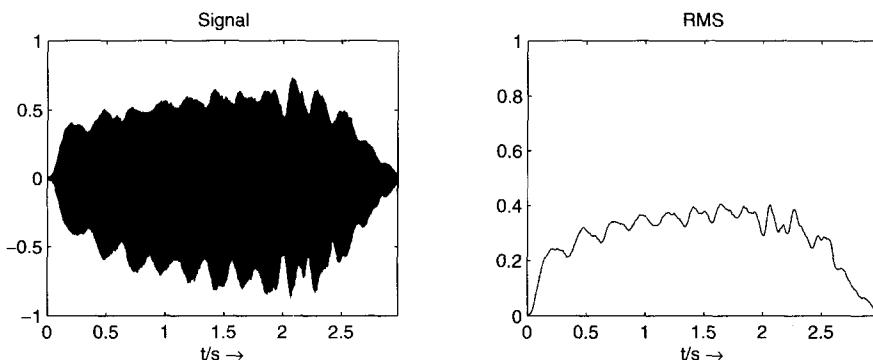


Figure 9.40 Signal and amplitude envelope (RMS value) of the signal.

can also be performed by a FIR filter, and in this case can be inserted into an FFT/IFFT-based analysis-synthesis scheme for a digital audio effect. The FFT window can be considered a lowpass FIR filter, and one of the reasons for the crucial choice of window size for a short-time Fourier transform is found in the separation between shape and matter: if the window is too short, the envelope will follow rapid oscillations which should not be included. If the window is too large, the envelope will not take into account tremolos which should be included. The following M-file 9.24 calculates the amplitude envelope of a signal according to an RMS algorithm.

```

M-file 9.24 (UX_rms.m)
% UX_rms.m
clear; clf
%----- USER DATA -----
[DAFx_in, FS] = wavread('x1.wav');
hop = 256; % hop size between two FFTs
WLen = 1024; % length of the windows
w = hanningz(WLen);
%----- some initializations -----
WLen2 = WLen/2;
normW = norm(w,2);
pft = 1;
lf = floor((length(DAFx_in) - WLen)/hop);
feature_rms = zeros(lf,1);
tic
=====
pin = 0;
pend = length(DAFx_in) - WLen;

while pin<pend
    grain = DAFx_in(pin+1:pin+WLen).* w;
    feature_rms(pft) = norm(grain,2) / normW;
    pft = pft + 1;
    pin = pin + hop;
end
%
toc
subplot(2,2,1); plot(DAFx_in); axis([1 pend -1 1])
subplot(2,2,2); plot(feature_rms); axis([1 lf -1 1])

```

There is another definition of the amplitude envelope of a signal: It could ideally be considered the magnitude of the analytical signal $x^+(n) = x(n) + j\hat{x}(n)$, where the real part is the input signal $x(n)$ and the imaginary part is the Hilbert transform of the real part (see section 4.2.3). However, this definition does not fit with the perception: this envelope is supposed to vary slowly with time and not include frequency information. So the magnitude of the analytical signal which, except for a sinusoidal signal, keeps oscillations following the fundamental frequency, does not fit the definition of an amplitude detector.

Center of Gravity of a Spectrum (Spectral Centroid)

An important feature of a sound is the evolution of the “richness of harmonics” over time. It has been clearly pointed out at the beginning of computer music that sounds synthesized with a fixed waveform give only static sounds, and that the sound’s harmonic content must evolve with time to give a lively sound impression. So algorithmic methods of synthesis have used this variation: additive synthesis

uses the balance between harmonics or partials, waveshaping or FM synthesis use an index which changes the richness by the strength of the components.

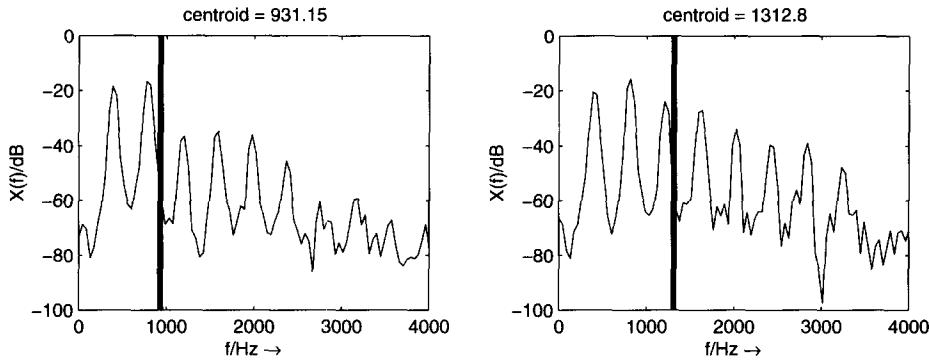


Figure 9.41 Center of gravity of a spectrum as a good indicator of the richness of a harmonic sound.

A good indication of the instantaneous richness of a sound can be measured by the center of gravity of its spectrum, as depicted in Fig. 9.41. A sound with a fixed pitch but with stronger harmonics has a higher center of gravity. It should be noted here that this center of gravity is linked to the pitch of the sound, and that this should be taken into account during the use of this feature. Thus, a good indicator of the instantaneous richness of a sound can be the ratio of the center of gravity divided by the pitch.

A straightforward method of calculating this centroid can be achieved inside an FFT/IFFT-based analysis-synthesis scheme. The spectral centroid is at the center of the spectral energy distribution and can be calculated by

$$\text{centroid} = \frac{\sum_{k=0}^{N/2} k \cdot |X(k)|}{\sum_{k=0}^{N/2} |X(k)|}. \quad (9.57)$$

The centroid is defined by the ratio of the sum of the magnitudes multiplied by the corresponding frequencies divided by the sum of the magnitudes and it is also possible to use the square $|X(k)|^2 = X(k)X^*(k)$ of the magnitudes.

Another method working in the time domain makes use of the property of the derivative of a sinusoid which gives $\frac{d}{dn} A_k \sin(\Omega_k n) = A_k \Omega_k \cdot \cos(\Omega_k n)$ with $\Omega_k = 2\pi \frac{f_k}{f_s}$. If we can express the input signal by a sum of sinusoids according to

$$x(n) = \sum_{k=0}^{N/2-1} A_k \sin(\Omega_k n), \quad (9.58)$$

the derivative of the input signal leads to

$$\frac{dx(n)}{dn} = \sum_{k=0}^{N/2-1} A_k \Omega_k \cos(\Omega_k n). \quad (9.59)$$

The spectral centroid can then be computed according to (9.57) by the ratio of the RMS value of the derivative of the input signal divided by the RMS value of the input signal itself. The derivative of the discrete-time input signal $x(n)$ can be approximated by $\Delta x(n) = x(n) - x(n - 1)$. The described time-domain method is quite effective because it does not need any FFT and is suitable for real-time applications. The following M-file 9.25 illustrates these possibilities.

```
M-file 9.25 (UX_centroid.m)
% UX_centroid.m

% feature_centroid1 and 2 are centroids
% calculate by two different methods
clear;clf
%---- USER DATA -----
[DAFx_in, FS] = wavread('x1.wav');
hop = 256; % hop size between two FFTs
WLen = 1024; % length of the windows
w = hanningz(WLen);
%---- some initializations -----
WLen2 = WLen/2;
tx = (1:WLen2+1)';
normW = norm(w,2);
coef = (WLen/(2*pi));
pft = 1;
lf = floor((length(DAFx_in) - WLen)/hop);
feature_rms = zeros(lf,1);
feature_centroid = zeros(lf,1);
feature_centroid2 = zeros(lf,1);
tic
%=====
pin = 0;
pend = length(DAFx_in) - WLen;

while pin<pend
    grain = DAFx_in(pin+1:pin+WLen).* w;
    feature_rms(pft) = norm(grain,2) / normW;
    f = fft(grain)/WLen2;
    fx = abs(f(tx));
    feature_centroid(pft) = sum(fx.*(tx-1)) / sum(fx);
    fx2 = fx.*fx;
    feature_centroid2(pft) = sum(fx2.*(tx-1)) / sum(fx2);
    grain2 = diff(DAFx_in(pin+1:pin+WLen+1)).* w;
    feature_deriv(pft) = coef * norm(grain2,2) / norm(grain,2);
    pft = pft + 1;
    pin = pin + hop;
end
```

```
% =====
toc
subplot(4,1,1); plot(feature_rms); xlabel('RMS')
subplot(4,1,2); plot(feature_centroid); xlabel('centroid 1')
subplot(4,1,3); plot(feature_centroid2); xlabel('centroid 2')
subplot(4,1,4); plot(feature_deriv); xlabel('centroid 3')
```

For each method the center of gravity is calculated in frequency bins. Figure 9.42 illustrates the results for each method. It can be seen at the end of a flute sound that the centroid parameter is very important: the variations of the centroid are quite independent of the RMS values of the signal. The centroid takes some time to oscillate and then is maintained until the end of the sound.

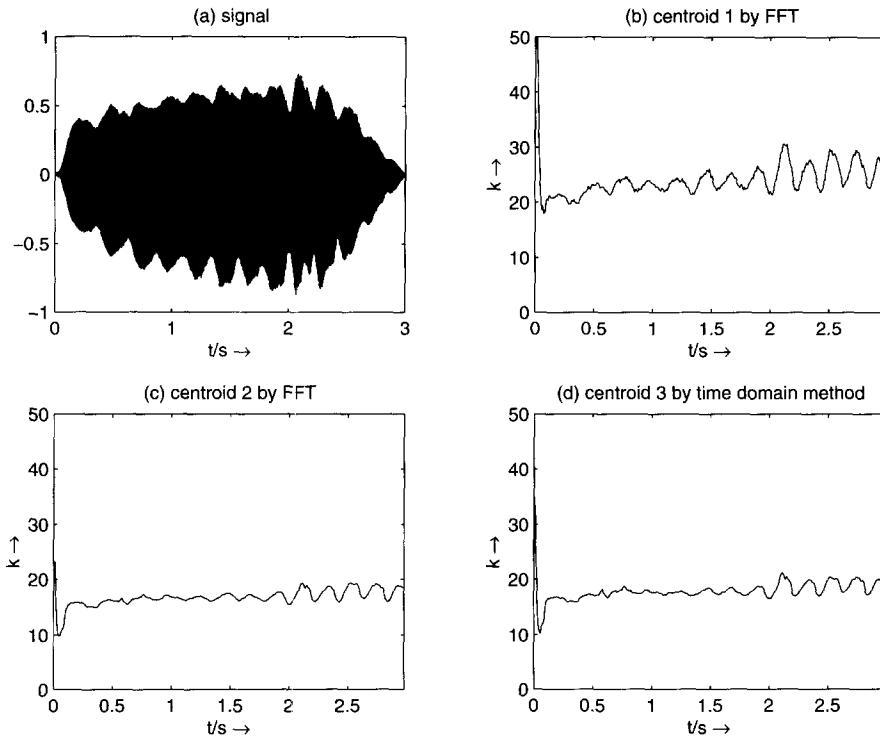


Figure 9.42 Spectral centroid computation in frequency and time domain. Plots (b)–(d) show the centroids in bins, where the corresponding tone pitch is given by $f_k = \frac{k}{N} f_s$ (with FFT length N and sampling frequency f_s).

A digital effect which relies on this feature is the mimicking of a natural sound by a synthetic one. As an example, we can use a waveshaping synthesis method. This method calculates a synthetic sound by distorting a sine wave with the help of a waveshaping function (also called nonlinear transfer function) and multiplying the result by an amplitude variation (see Fig. 9.43). The waveshaping function is

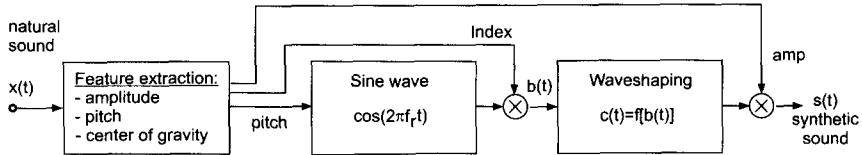


Figure 9.43 Mimicking with waveshaping.

usually a polynomial, because this function can be calculated with the aim of having a fixed output spectrum [Arf79, Bru79]. So apart from the pitch of the sine wave, this method relies on the evolution of two parameters: the amplitude of the sine wave, which is called the “index” because the sine wave is used as an input to the waveshaping function, and an amplitude factor, which is used as an amplitude envelope. If we extract the centroid and the RMS evolution from a natural sound, as well as the pitch, we can compute an index proportional to the ratio of the centroid towards the pitch (this proportional factor, apart from a necessary normalization, drives the general brightness of the sound) and an amplitude factor proportional to the extracted RMS feature. Thus we obtain a synthetic sound that retains some characteristics of the initial sound and is given by

$$\begin{aligned}
 s(t) &= \text{amp} \cdot f[\text{index} \cdot \cos(2\pi f_r t)] \\
 f_r &= \text{pitch}, \text{index} = \text{coef} \cdot \frac{\text{centroid}}{\text{pitch}}, \text{amp} = \text{rms}.
 \end{aligned} \tag{9.60}$$

The mimicking is improved when the waveshaping function is calculated for the spectrum at one point of the initial sound. Two other further improvements can be added: an amplitude normalization factor due to the fact that the index should only change the centroid and not the amplitude, and a correcting factor for the index due to the fact that the index and centroid of the synthetic sound have no reason to be in a linear relationship. But even the simple process we have described gives a variety of allotropic sounds which all resemble the original in some way, but are purely harmonic and do not contain any noisy components.

Autocorrelation Features

We can extract important features from the autocorrelation sequence of a windowed signal: an estimation of the harmonic/non-harmonic content of a signal, the odd/even harmonics ratio in the case of harmonic sounds and a voiced/unvoiced part in a speech signal. Several algorithms which determine whether a speech frame is voiced or unvoiced are known from speech research [Hes83]. Voiced/unvoiced detection is used either for speech recognition or for synthesis. For digital audio effects, such a feature is useful as a control parameter for an adaptive audio effect. The first peak value of the normalized autocorrelation sequence $r_{xx}(m)$ for $m > 0$ is a good indicator of the unvoiced or voiced part of a signal, as shown in Fig. 9.44. When sounds are harmonic, the first autocorrelation peak ($m > 0$) on the abscissa corresponds to the pitch period of this sound. The value of this peak will be maximum

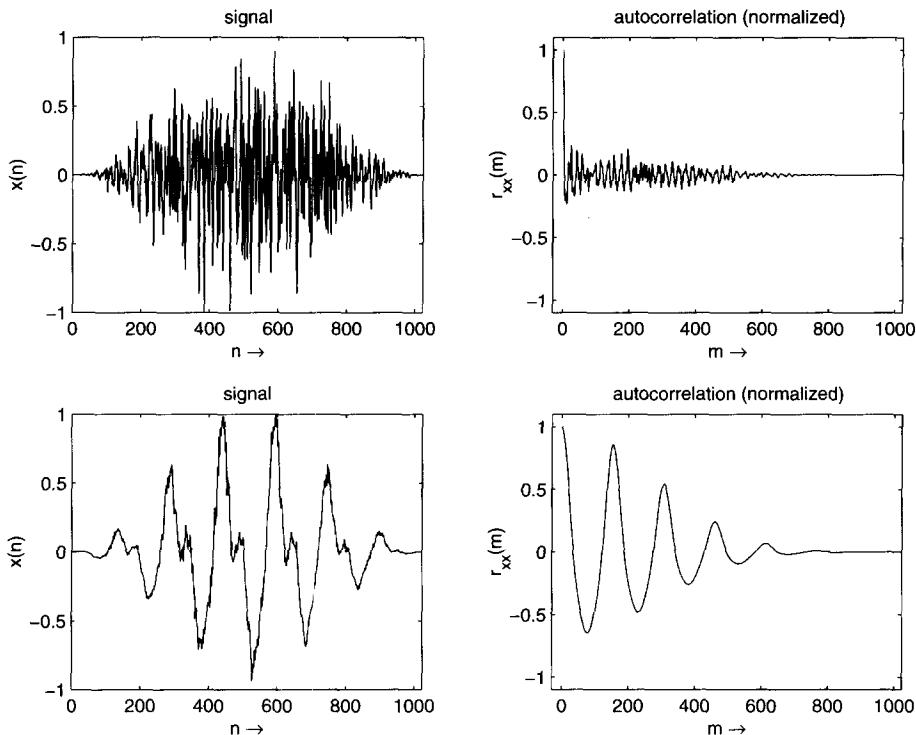


Figure 9.44 Unvoiced (upper part) and voiced (lower part) signals and the corresponding autocorrelation sequence $r_{xx}(m)$.

if the sound is harmonic and minimum if the sound is noisy. If a window is used, which gives a better estimate for pitch extraction, this peak value will not go to one but will be weighted by the autocorrelation of the two windows. This first peak value will be denoted pv and is a good indication of voiced/noisy parts in a spoken or sung voice. In the case of harmonic sounds, it can be noted that the odd/even harmonics ratio can also be retrieved from the value at half of the time lag of the first peak.

An alternative computation of the autocorrelation sequence can be performed in the frequency domain [OS75]. Normally, the autocorrelation is computed from the power spectrum $|X(k)|^2$ of the input signal by $r_{xx}(m) = \text{IFFT}[|X(k)|^2]$. Here, we perform the IFFT of the magnitude $|X(k)|$ (square root of the power spectrum), which is computed from the FFT of a windowed signal. This last method is illustrated by the following M-file 9.26 that leads to a curve following the voiced/unvoiced feature, as shown in Fig. 9.45.

M-file 9.26 (UX_voiced.m)

```
% UX_voiced.m
```

```
% feature_voice is a measure of the maximum of the second peak
% of the acf
clear;clf
%---- USER DATA -----
[DAFx_in, FS] = wavread('x1.wav');
hop = 256; % hop size between two FFTs
WLen = 1024; % length of the windows
w = hanningz(WLen);
%---- some initializations -----
WLen2 = WLen/2;
tx = (1:WLen2+1)';
normW = norm(w,2);
coef = (WLen/(2*pi));
pft = 1;
lf = floor((length(DAFx_in) - WLen)/hop);
feature_voiced = zeros(lf,1);
tic
%=====
pin = 0;
pend = length(DAFx_in) - WLen;

while pin<pend
    grain = DAFx_in(pin+1:pin+WLen).* w;
    f = fft(grain)/WLen2;
    f2 = real(ifft(abs(f)));
    f2 = f2/f2(1);
    [v,i1] = min(f2(1:WLen2)>0.);
    f2(1:i1) = zeros(i1,1);
    [v,imax] = max(f2(1:WLen2));
    feature_voiced(pft) = v;
    pft = pft + 1;
    pin = pin + hop;
end
% =====
toc
subplot(2,1,1)
plot(feature_voiced)
```

A particular way to use this feature is the construction of an adaptive time stretching effect, where the stretching ratio α depends on this feature according to a mapping function $\alpha = 8^{pv}$ (see Fig. 9.46). The time stretching ratio will vary from 1 to 8, depending on the evolution of pv over time. A threshold detector can help to force this ratio to one in the case of silence. This leads to great improvements over regular time stretching algorithms.

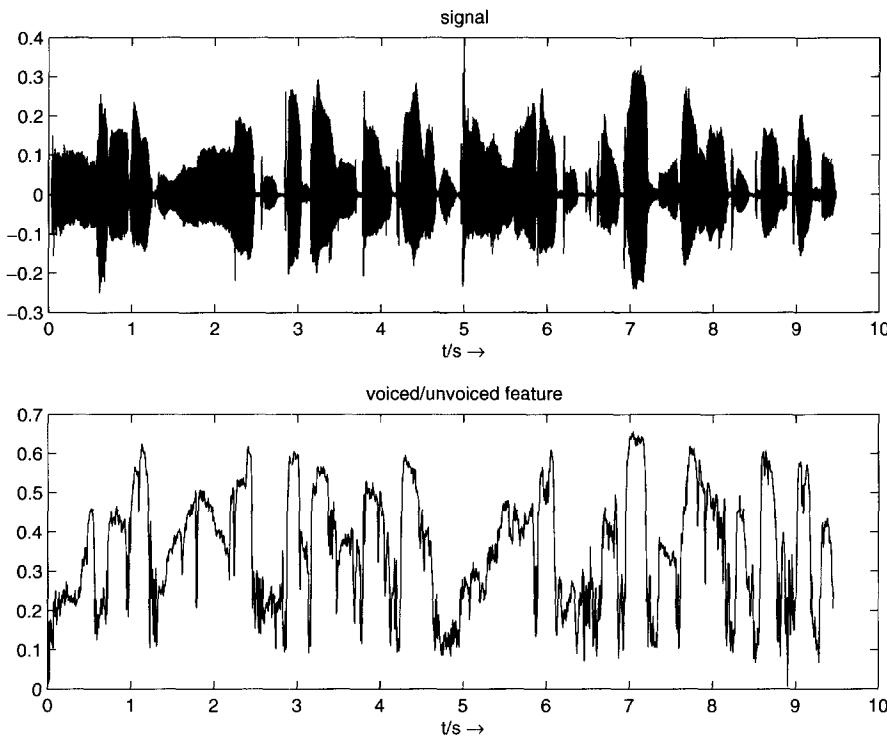


Figure 9.45 Vocal signal and the “voiced/unvoiced” feature $pv(n)$.

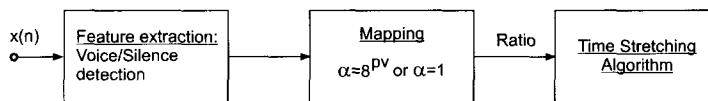


Figure 9.46 Adaptive time stretching based on autocorrelation feature.

Statistical Features

As an example, Dubnov [DT96] has obtained the classification of instrumental timbres on a 2-D map by using skew and kurtosis. These features can help in defining a texturization of a sound. The texture of a sound is very difficult to evaluate: why a trumpet does not sound like a string does not rely only on a spectral richness. The way the individual components are synchronized or not is an important key for defining a texture. Many other features can be extracted from a sound [HB98, RDS⁺99, DH00], just to mention a few from a very active field of research.

9.5 Conclusion

The central topic of this chapter is the division of the audio signal into its source signal and a time-varying filter derived from the spectral envelope of the signal. These two features are individually processed before synthesis of an output signal. The source-filter model of an audio signal, originally a basic technique for speech processing, allows the implementation of several digital audio effects based on these two global features of an audio signal and opens up new vistas for experimentation and further research. These global features can either be extracted by time-frequency techniques (FFT/IFFT) and the cepstrum method or time-domain techniques based on linear prediction (LPC). Both techniques deliver a source-filter model of the audio input signal. Beyond it, they allow the extraction of further global features such as pitch or fundamental frequency, which can be estimated by the cepstrum method or autocorrelation techniques applied to the input directly or the extracted source signal. Further global features such as amplitude envelope, spectral centroid, and autocorrelation features (voiced/unvoiced detection) have been introduced, which can be estimated by simple time-domain or by advanced time-frequency techniques. The main objective here is the introduction and estimation of these parameters for the control of various other digital audio effects, which are presented throughout this book. A further alternative to the source-filter processing presented in this chapter, is the separation of the audio signal into individual components such as sinusoids and noise, which is discussed in Chapter 10.

Bibliography

- [AKZ99] R. Althoff, F. Keiler, and U. Zölzer. Extracting Sinusoids from Harmonic Signals. In *Proc. DAFX-99 Digital Audio Effects Workshop*, pp. 97–100, Trondheim, December 1999.
- [Arf79] D. Arfib. Digital synthesis of complex spectra by means of multiplication of nonlinear distorted sine waves. *J. Audio Eng. Soc.*, 27:757–768, October 1979.
- [Bru79] M. Le Brun. Digital waveshaping synthesis. *J. Audio Eng. Soc.*, 27:250–265, April 1979.
- [Can98] P. Cano. Fundamental frequency estimation in the SMS analysis. In *Proc. DAFX-98 Digital Audio Effects Workshop*, pp. 99–102, Barcelona, November 1998.
- [DH00] M. Desainte-Catherine and P. Hanna. Statistical approach for sounds modeling. In *Proc. DAFX-00 Conference on Digital Audio Effects*, pp. 91–96, Verona, December 2000.
- [DM00] M. Desainte-Catherine and S. Marchand. High-precision Fourier analysis of sounds using signal derivatives. *J. Audio Eng. Soc.*, 48(7/8):654–667, July/August 2000.

- [DT96] S. Dubnov and N. Tishby. Testing for gaussianity and non-linearity in the sustained portion of musical sounds. In *Proc. of the Journées Informatique Musicale*, 1996.
- [HB98] P. Herrera and J. Bonada. Vibrato extraction and parameterization in the spectral modeling synthesis framework. In *Proc. DAFX-98 Digital Audio Effects Workshop*, pp. 107–110, Barcelona, November 1998.
- [Hes83] W. Hess. *Pitch Determination of Speech Signals*. Springer-Verlag, 1983.
- [HKS⁺00] A. Härma, M. Karjalainen, L. Savioja, V. Välimäki, U.K. Laine, and J. Huopaniemi. Frequency-warped signal processing for audio applications. *J. Audio Eng. Soc.*, 48(11):1011–1031, 2000.
- [JN84] N.S. Jayant and P. Noll. *Digital Coding of Waveforms*. Prentice-Hall, 1984.
- [KA90] P. Kroon and B.S. Atal. Pitch predictors with high temporal resolution. In *Proceedings of the ICASSP*, pp. 661–664, Albuquerque, 1990.
- [KAZ00] F. Keiler, D. Arfib, and U. Zölzer. Efficient linear prediction for digital audio effects. In *Proc. DAFX-00 Conference on Digital Audio Effects*, pp. 19–24, Verona, December 2000.
- [LS81] P. Lansky and K. Steiglitz. Synthesis of timbral families by warped linear prediction. *Computer Music Journal*, 5(3):45–47, 1981.
- [LVKL96] T.I. Laakso, V. Välimälki, M. Karjalainen, and U.K. Laine. Splitting the unit delay. *IEEE Signal Processing Magazine*, 13:30–60, 1996.
- [Mak75] J. Makhoul. Linear prediction: A tutorial review. *Proceedings of the IEEE*, 63(4):561–580, 1975.
- [Mak77] J. Makhoul. Stable and efficient lattice methods for linear prediction. *IEEE Trans. on Acoustics, Speech, and Signal Processing*, ASSP-25(5):423–428, October 1977.
- [Mar72] J.D. Markel. The SIFT algorithm for fundamental frequency estimation. *IEEE Trans. on Audio and Electroacoustics*, 20(5):367–377, 1972.
- [Mar98] S. Marchand. Improving spectral analysis precision with enhanced phase vocoder using signal derivatives. In *Proc. DAFX-98 Digital Audio Effects Workshop*, pp. 114–118, Barcelona, November 1998.
- [Mar00] S. Marchand. *Sound Models for Computer Music*. PhD thesis, University of Bordeaux, October 2000.
- [MG76] J.D. Markel and A.H. Gray. *Linear Prediction of Speech*. Springer-Verlag, 1976.
- [Moo79] J. A. Moorer. The use of linear prediction of speech in computer music applications. *J. Audio Eng. Soc.*, 27(3):134–140, March 1979.

- [Nol64] A.M. Noll. Short-time spectrum and “cepstrum” techniques for vocal-pitch detection. *J. Acoust. Soc. Am.*, 36(2):296–302, 1964.
- [Orf90] S.J. Orfanidis. *Optimum Signal Processing, An Introduction*. McGraw-Hill, 2nd edition, 1990.
- [OS75] A.V. Oppenheim and R.W. Schafer. *Digital Signal Processing*. Prentice-Hall, 1975.
- [O'S00] D. O'Shaugnessy. *Speech Communication*. Addison-Wesley, 2nd edition, 2000.
- [PM96] J.G. Proakis and D.G. Manolakis. *Digital Signal Processing*. Prentice-Hall, 1996.
- [RDS⁺99] S. Rossignol, P. Depalle, J. Soumagne, X. Rodet, and J.-L. Colette. Vibrato: detection, estimation, extraction, modification. In *Proc. DAFX-99 Digital Audio Effects Workshop*, pp. 175–179, Trondheim, December 1999.
- [RS78] L.R. Rabiner and R.W. Schafer. *Digital Processing of Speech Signals*. Prentice-Hall, 1978.
- [Sch66] P. Schaeffer. *Traité des objets musicaux*. Seuil, 1966.
- [Sch68] M.R. Schroeder. Period histogram and product spectrum: New methods for fundamental-frequency measurement. *J. Acoust. Soc. Am.*, 43(4):829–834, 1968.
- [Sch99] M.R. Schroeder. *Computer Speech*. Springer-Verlag, 1999.
- [Son68] M.M. Sondhi. New methods of pitch extraction. *IEEE Trans. on Audio and Electroacoustics*, 16(2):262–266, 1968.
- [Str80] H.W. Strube. Linear prediction on a warped frequency scale. *J. Acoust. Soc. Am.*, 68(4):1071–1076, October 1980.

Chapter 10

Spectral Processing

X. Amatriain, J. Bonada, A. Loscos, X. Serra

10.1 Introduction

In the context of this book, we are looking for representations of sound signals and signal processing systems that can provide ways to design sound transformations in a variety of music applications and contexts. It should have been clear throughout the book that several points of view have to be considered, including a mathematical, thus objective perspective, and a cognitive, thus mainly subjective, standpoint. Both points of view are necessary to fully understand the concept of sound effects and to be able to use the described techniques in practical situations.

The mathematical and signal processing points of view are straightforward to present, although not necessarily easy, since the language of the equations and of flow diagrams is suitable for them. However, the top-down implications are much harder to express due to the huge number of variables involved and to the inherent perceptual subjectivity of the music making process. This is clearly one of the main challenges of the book and the main reason for its existence.

The use of a spectral representation of a sound yields a perspective that is sometimes closer to the one used in a sound engineering approach. By understanding the basic concepts of frequency domain analysis, we are able to acquire the tools to use a large number of effects processors and to understand many types of sound transformation systems. Moreover, as the frequency domain analysis is a somewhat similar process to the one performed by the human hearing system, it yields fairly intuitive intermediate representations.

The basic idea of spectral processing is that we can analyze a sound to obtain alternative frequency domain representations, which can then be transformed and inverted to produce new sounds (see Fig. 10.1). Most of the approaches start by developing an analysis/synthesis system from which the input sound is reconstructed

without any perceptual loss of sound quality. The techniques described in Chapters 8 and 9 are clear examples of this approach. Then the main issue is what the intermediate representation is and what parameters are available to apply the desired transformations.

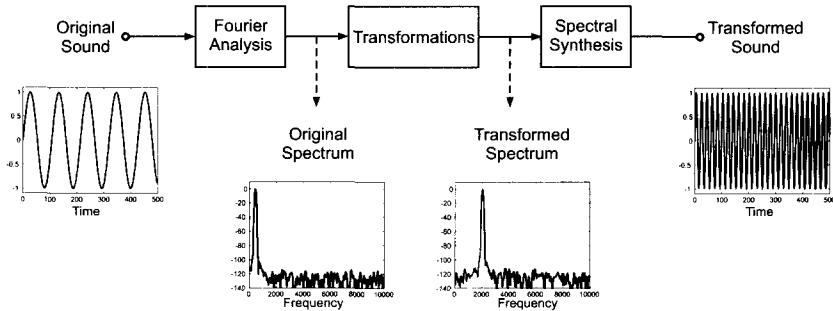


Figure 10.1 Block diagram of a simple spectral processing framework.

Perceptual or musical concepts such as timbre or pitch are clearly related to the spectral characteristics of a sound. Even some very common processes for sound effects are better explained using a frequency domain representation. We usually think about the frequency axis when we talk about equalizing, filtering, pitch shifting, harmonizing ... In fact, some of them are specific to this signal processing approach and do not have an immediate counterpart in the time domain. On the other hand, most (but not all) of the sound effects presented in this book can be implemented in the frequency domain.

Another issue is whether or not this approach is the most efficient, or practical, for a given application. The process of transforming a time domain signal into a frequency domain representation is, by itself, not an immediate step. Some parameters are difficult to adjust and force us to take several compromises. Some settings, such as the size of the analysis window, have little or nothing to do with the high-level approach we intend to favor, and require the user to have a basic signal processing understanding.

In that sense, when we talk about higher level spectral processing we are thinking of an intermediate analysis step in which relevant features are extracted or computed from the spectrum. These relevant features should be much closer to a musical or high-level approach. We can then process the features themselves or even apply transformations that keep the features unchanged. For example, we can extract the fundamental frequency and the spectral shape from a sound and then modify the fundamental frequency without affecting the shape of the spectrum.

Assuming that there is no single representation and processing system optimal for everything, our approach will be to present a set of complementary spectral models that can be combined to be used for the largest possible set of sounds and musical applications.

In section 10.2 we introduce two spectral models: *sinusoidal* and *sinusoidal plus residual*. These models already represent a step up on the abstraction ladder and

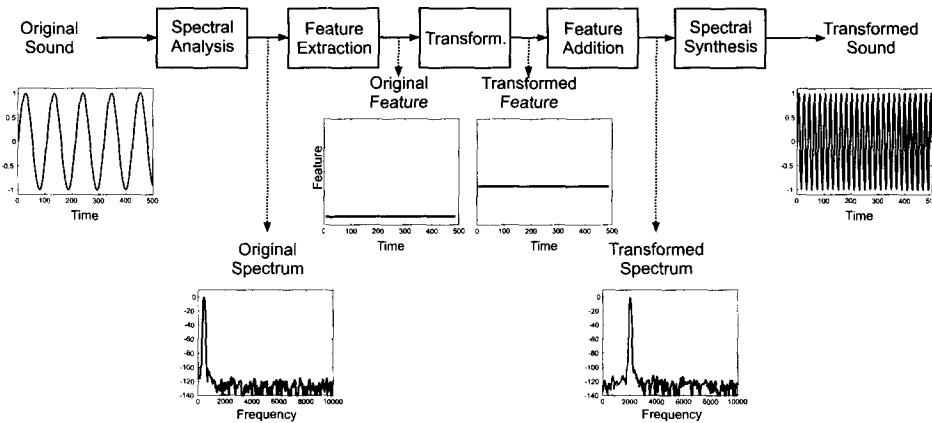


Figure 10.2 Block diagram of a higher-level spectral processing framework.

from either of them, we can identify and extract higher-level information of a sound, such as: harmonics, pitch, spectral shape, vibrato, or note boundaries, that is higher level features. This analysis step brings the representation closer to our perceptual understanding of a sound. The complexity of the analysis will depend on the type of feature that we wish to identify and the sound to analyze. The benefits of going to this higher level of analysis are enormous and open up a wide range of new musical applications.

Having set the basis of the sinusoidal plus residual model, we will then give some details of the techniques used both in its analysis and synthesis process, providing MATLAB code to implement an analysis-synthesis framework in section 10.3. This MATLAB implementation is based on the *spectral modeling synthesis* (SMS) framework [SMS]. SMS is a set of spectral-based techniques and related implementations for the analysis/transformation/synthesis of an audio signal based on the scheme presented in Fig. 10.2.

In section 10.4 we will provide a set of basic audio effects and transformations based on the implemented sinusoidal plus residual analysis/synthesis. MATLAB code is provided for all of them.

We will finish with an explanation of content-dependent processing implementations. In section 10.5.1 we introduce a real-time singing voice conversion application that has been developed for use in karaoke, and in section 10.5.2 we define the basis of a nearly lossless time scaling algorithm. The complexity and extension of these implementations prevent us from providing the associated MATLAB code, so we leave that task as a challenge for advanced readers.

10.2 Spectral Models

The most common approach for converting a time domain signal into its frequency domain representation is the short-time Fourier transform (STFT). It is a general

technique from which we can implement lossless analysis/synthesis systems. Many sound transformation systems are based on direct implementations of the basic algorithm and several examples have been presented in Chapter 8.

In this chapter, we will briefly mention the sinusoidal model and will concentrate, with a MATLAB sample code, on the sinusoidal plus residual model. The decision as to what spectral representation to use in a particular situation is not an easy one. The boundaries are not clear and there are always compromises to take into account, such as: (1) sound fidelity, (2) flexibility, (3) coding efficiency, and (4) computational requirements. Ideally, we want to maximize fidelity and flexibility while minimizing memory consumption and computational requirements. The best choice for maximum fidelity and minimum computation time is the STFT that, anyhow, yields a rather inflexible representation and inefficient coding scheme. Thus our interest in finding higher-level representations as the ones we present in this section.

10.2.1 Sinusoidal Model

Using the output of the STFT, the sinusoidal model represents a step towards a more flexible representations while compromising both sound fidelity and computing time. It is based on modeling the time-varying spectral characteristics of a sound as sums of time-varying sinusoids. The input sound $s(t)$ is modeled by

$$s(t) = \sum_{r=1}^R A_r(t) \cos[\theta_r(t)] \quad (10.1)$$

where $A_r(t)$ and $\theta_r(t)$ are the instantaneous amplitude and phase of the r^{th} sinusoid, respectively [MQ86, SS87].

To obtain a sinusoidal representation from a sound, an analysis is performed in order to estimate the instantaneous amplitudes and phases of the sinusoids. This estimation is generally done by first computing the STFT of the sound, as described in Chapter 8, then detecting the spectral peaks (and measuring the magnitude, frequency and phase of each one), and finally organizing them as time-varying sinusoidal tracks.

It is a quite general technique that can be used in a wide range of sounds and offers a gain in flexibility compared with the direct STFT implementation.

10.2.2 Sinusoidal plus Residual Model

The sinusoidal plus residual model can cover a wide “compromise space” and can in fact be seen as the generalization of both the STFT and the sinusoidal models. Using this approach, we can decide what part of the spectral information is modeled as sinusoids and what is left as STFT. With a good analysis, the sinusoidal plus residual representation is very flexible while maintaining a good sound fidelity, and the representation is quite efficient. In this approach, the sinusoidal representation is

used to model only the stable partials of a sound. The residual, or its approximation, models what is left, which should ideally be a stochastic component. This model is less general than either the STFT or the sinusoidal representations but it results in an enormous gain in flexibility [Ser89, SS90, Ser96].

The input sound $s(t)$ is modeled by

$$s(t) = \sum_{r=1}^R A_r(t) \cos[\theta_r(t)] + e(t) \quad (10.2)$$

where $A_r(t)$ and $\theta_r(t)$ are the instantaneous amplitude and phase of the r^{th} sinusoid, respectively, and $e(t)$ is the noise component at time t (in seconds).

The sinusoidal plus residual model assumes that the sinusoids are stable partials of the sound with a slowly changing amplitude and frequency. With this restriction, we are able to add major constraints to the detection of sinusoids in the spectrum and omit the detection of the phase of each peak. The instantaneous phase that appears in the equation is taken to be the integral of the instantaneous frequency $\omega_r(t)$, and therefore satisfies

$$\theta_r(t) = \int_0^t \omega_r(\tau) d\tau \quad (10.3)$$

where $\omega(t)$ is the frequency in radians, and r is the sinusoid number. When the sinusoids are used to model only the stable partials of the sound, we refer to this part of the sound as the deterministic component.

Within this model we can either leave the residual signal, $e(t)$, to be the difference between the original sound and the sinusoidal component, resulting into an identity system, or we can assume that $e(t)$ is a stochastic signal. In this case, the residual can be described as filtered white noise,

$$e(t) = \int_0^t h(t, \tau) u(\tau) d\tau \quad (10.4)$$

where $u(t)$ is white noise and $h(t, \tau)$ is the response of a time varying filter to an impulse at time t . That is, the residual is modeled by the time-domain convolution of white noise with a time-varying frequency-shaping filter.

The implementation of the analysis for the sinusoidal plus residual model is more complex than the one for the sinusoidal model. Figure 10.3 shows a simplified block diagram of this analysis.

The first few steps are the same as those in a sinusoidal-only analysis. The major differences start in the peak continuation process since in order to have a good partial-residual decomposition we have to refine this peak-continuation process in such a way as to be able to identify the stable partials of the sound. Several strategies can be used to accomplish this. The simplest case is when the sound is monophonic and pseudo-harmonic. By using the fundamental frequency information in the peak continuation algorithm, we can easily identify the harmonic partials.

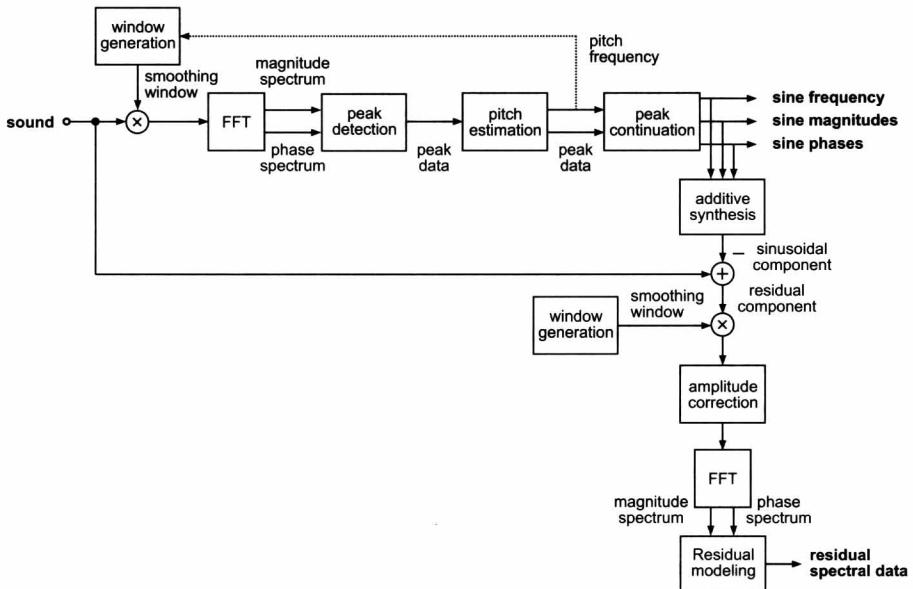


Figure 10.3 Block diagram of the sinusoidal plus residual analysis.

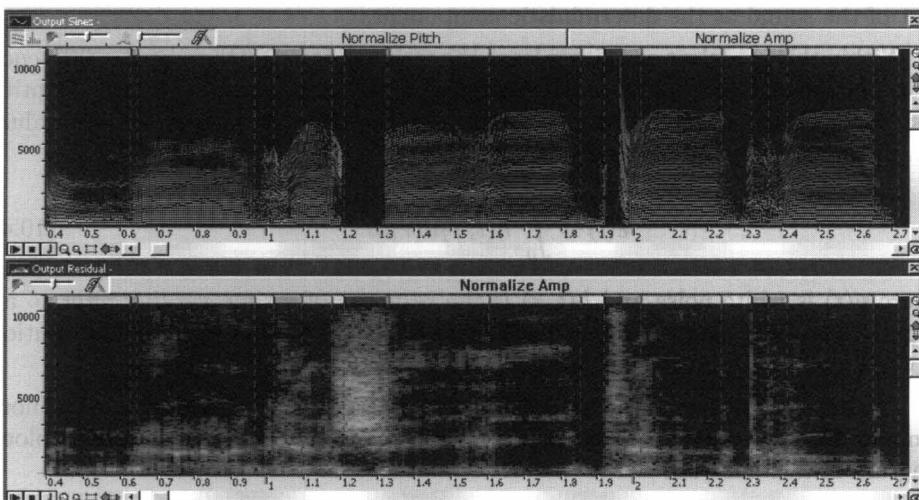


Figure 10.4 Spectrogram of sinusoidal component (upper plot) and residual component (lower plot).

The residual component is obtained by first generating the sinusoidal component with additive synthesis, and then subtracting it from the original waveform. This is possible because the instantaneous phases of the original sound are matched and therefore the shape of the time domain waveform is preserved. A spectral analysis of this time domain residual is done by first windowing it, using a window which is independent of the one used to find sinusoids, and thus we are free to choose a different time-frequency compromise. An amplitude correction step can improve the time smearing produced in the sinusoidal subtraction. Then the FFT is computed and the resulting spectrum can be modeled using several existing techniques. The spectral phases might be discarded if the residual can be approximated as a stochastic signal. Figure 10.4 shows a spectrogram illustrating the sinusoidal and residual components.

The original sinusoidal plus residual model has led to other different spectral models that still share some of its basic principles [DQ97, FHC00, VM00].

10.3 Techniques

It is beyond the scope of this chapter to discuss deeply the whole analysis-synthesis process that results in a sinusoidal plus residual representation of the sound, but let us describe in some detail the major steps.

10.3.1 Analysis

The analysis step of the sinusoidal plus residual model has already been presented in the previous section and is illustrated in Fig. 10.3. Next we will introduce the most important techniques and the basic considerations that need be taken into account when analyzing a sound.

Previous Considerations: STFT Settings

In this section, we will see that the STFT process is far from being unsupervised, and its settings are indeed critical in order to get a good representation of the sound. The main parameters involved in this step are window size, window type, frame size and hop size.

As has already been mentioned in previous chapters, the first step involved in the process of converting a time domain signal into its frequency domain representation, is the windowing of the sound. This operation involves selecting a number of samples from the sound signal and multiplying their value by a windowing function [Har78].

The number of samples taken in every processing step is defined by the window size. It is a crucial parameter, especially if we take into account that the number of spectral samples that the DFT will yield at its output corresponds to half the number of samples of its input spread over half of the original sampling rate. We will not go into the details of the DFT mathematics that lead to this property, but it is

very important to note that the longer the window, the more frequency resolution we will have. On the other hand, it is straightforward to see the drawback of taking very long windows: the loss of time resolution. This phenomenon is known as the time vs. frequency resolution trade-off (see Fig. 10.5). A more specific limitation of the window size has to do with choosing windows with odd sample-length in order to guarantee even symmetry about the origin.

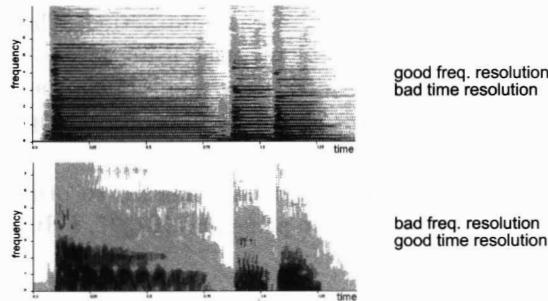


Figure 10.5 Time vs. frequency resolution trade-off.

The kind of window used also has a very strong effect on the qualities of the spectral representation we will obtain. At this point we should remember that a time domain multiplication (such as the one between the signal and the windowing function) becomes a frequency domain convolution between the Fourier transforms of each of the signals (see Fig. 10.6). One may be tempted to forget about deciding on these matters and apply no window at all, just taking n samples from the signal and feeding them into the chosen FFT algorithm. Even in this case, though, a rectangular window is being used, so the spectrum of the signal is being convolved with the transform of a rectangular pulse, a *sinc-like* function.

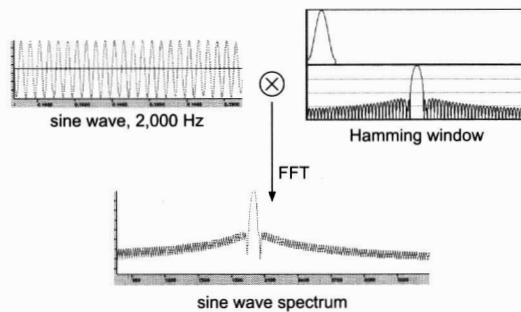


Figure 10.6 Effect of applying a window in the time domain.

Two features of the transform of the window are specially relevant to whether a particular function is useful or not: the width of the main lobe, and the main to highest side lobe relation. The main lobe bandwidth is expressed in bins (spectral samples) and, in conjunction with the window size, defines the ability to distinguish

two sinusoidal peaks (see Fig. 10.7). The following formula expresses the relationship the window size M , the main lobe bandwidth B_s and the sampling rate f_s should have in order to distinguish two sinusoids of frequency f_k and f_{k+1} :

$$M \geq B_s \frac{f_s}{|f_{k+1} - f_k|}. \quad (10.5)$$

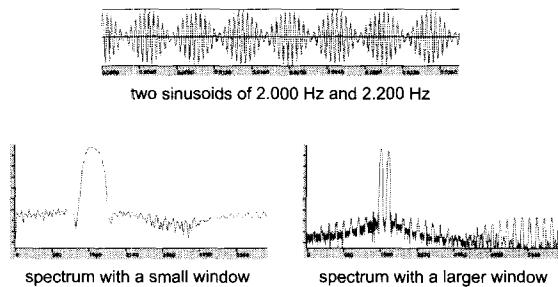


Figure 10.7 Effect of the window size in distinguishing between two sinusoids.

The amplitude relationship between the main and the highest side lobe explains the amount of distortion a peak will receive from surrounding partials. It would be ideal to have a window with an extremely narrow main lobe and a very high main to secondary lobe relation. However, the inherent trade-off between these two parameters forces a compromise to be taken.

Common windows that can be used in the analysis step are Rectangular, Triangular, Kaiser-Bessel, Hamming, Hanning and Blackmann-Harris. In the code supplied in this chapter, we have chosen a Blackmann-Harris 92 dB window for the sake of simplicity. This window has a rather wide main lobe (9 bins) but an extremely high main-to-secondary lobe relation of 92 dB. This difference is so close to the dynamic range of a 16-bit representation that, in that case, we need only take into account the influence of the main lobe. The following M-file 10.1 implements the generation of a Blackman-Harris window.

```
M-file 10.1 (bh92.m)
function [bh92SINE2SINE, bh92SINE2SINESize] = bh92SINE2SINEgeneration;
%function [bh92SINE2SINE, bh92SINE2SINESize] = bh92SINE2SINEgeneration;
%
% ==> generation of the Blackman-Harris window
% output data:
%     bh92SINE2SINESize: size of the window
%     bh92SINE2SINE: (sampled) window

bh92SINE2SINESize = 4096;
bh92SINE2SINE = zeros(bh92SINE2SINESize, 1);
bh92N = 512;
bh92const = [.35875, .48829, .14128, .01168];
```

```

bh92Theta = -4*2*pi/bh92N;
bh92ThetaIncr = 8*2*pi/bh92N/bh92SINE2SINESize;
for i=1:bh92SINE2SINESize
    for m=0:3
        bh92SINE2SINE(i)=bh92SINE2SINE(i)-bh92const(m+1)/2*...
            (sine2sine(bh92Theta-m*2*pi/bh92N,bh92N)+...
            sine2sine(bh92Theta+m*2*pi/bh92N,bh92N));
    end;
    bh92Theta = bh92Theta + bh92ThetaIncr;
end;
bh92SINE2SINE = bh92SINE2SINE/bh92SINE2SINE(bh92SINE2SINESize/2+1);

```

The value of the `sine2sine` function (not included in the basic MATLAB package) is computed as follows:

```

M-file 10.2 (sine2sine.m)
function x = sine2sine( x , N )
% sine2sine function !!!
x = sin((N/2)*x) / sin(x/2);

```

One may think that a possible way of overcoming the time/frequency trade-off is to add zeros to the windowed signals in order to have a longer FFT and so increase the frequency resolution. This process is known as *zero-padding* and it represents an interpolation in the frequency domain. Thus, when we zero-pad a signal before the DFT process, we are not adding any information to its frequency representation (we will still not distinguish two sinusoids if (10.5) is not satisfied), but we are indeed increasing the frequency resolution by adding intermediate interpolated bins. This process can help in the peak detection process, as explained later.

A final step is the circular shift already described in section 8.2.2. This *buffer centering* guarantees the preservation of zero-phase conditions in the analysis process.

Once the spectrum of a *frame* has been computed, the window must move to the next position in the waveform in order to take the next set of samples. The distance between the centers of two consecutive windows is known as *hop size*. If the hop size is smaller than the window size, we will be including some overlap, that is, some samples will be used more than once in the analysis process. In general, the more overlap, the smoother the transitions of the spectrum will be across time, but that is a computationally expensive process. The window type and the hop size must be chosen in such a way that the resulting envelope adds approximately to a constant, following the equation

$$A_w(m) \equiv \sum_{n=-\infty}^{\infty} w(m - nH) \approx \text{constant}. \quad (10.6)$$

A measure of the deviation of A_w from a constant is the difference between the maximum and minimum values for the envelope as a percentage of the maximum

value:

$$d_w = 100 \times \frac{\max_m[A_w(m)] - \min_m[A_w(m)]}{\max_m[A_w(m)]} \quad (10.7)$$

This measure is referred to as the *amplitude deviation* of the overlap factor. Variables should be chosen accordingly to keep this factor around or below 1 percent.

We have seen that the STFT process that is bound to provide a suitable frequency domain representation of the input signal, is a far from trivial process and is dependent on some low-level parameters closely related to the signal processing domain. A little theoretical knowledge is required but only practice will surely lead to the desired results.

Peak Detection

The sinusoidal model assumes that each spectrum of the STFT representation can be explained by a series of sinusoids. For a given frequency resolution, using enough points in the spectrum, a sinusoid can be identified by its shape. Theoretically, a sinusoid that is stable both in amplitude and in frequency - a partial - has a well-defined frequency representation: the transform of the analysis window used to compute the Fourier transform. It should be possible to take advantage of this characteristic to distinguish partials from other frequency components. However, in practice this is rarely the case, since most natural sounds are not perfectly periodic and do not have nicely spaced and clearly defined peaks in the frequency domain. There are interactions between the different components, and the shapes of the spectral peaks cannot be detected without tolerating some mismatch. Only certain instrumental sounds (e.g., the steady-state part of an oboe sound) are periodic enough and sufficiently free from prominent noise components that the frequency representation of a stable sinusoid can be recognized easily in a single spectrum (see Fig. 10.8). A practical solution is to detect as many peaks as possible, with some small constraints, and delay the decision of what is a “well behaved” partial, to the next step in the analysis: the peak continuation algorithm.

A “peak” is defined as a local maximum in the magnitude spectrum, and the only practical constraints to be made in the peak search are to have a frequency range and a magnitude threshold. Due to the sampled nature of the spectrum returned by the FFT, each peak is accurate only to within half a sample. A spectral sample represents a frequency interval of f_s/N Hz, where f_s is the sampling rate and N is the FFT size. Zero-padding in the time domain increases the number of spectral samples per Hz and thus increases the accuracy of the simple peak detection (see previous section). However, to obtain frequency accuracy on the level of 0.1 percent of the distance from the top of an ideal peak to its first zero crossing (in the case of a Rectangular window), the zero-padding factor required is 1000.

A more efficient spectral interpolation scheme is to zero-pad such that quadratic (or other simple) spectral interpolation, using only samples immediately surrounding the maximum-magnitude sample, suffices to refine the estimate to 0.1 percent accuracy. That is the approach we have chosen and is illustrated in Fig. 10.9. The

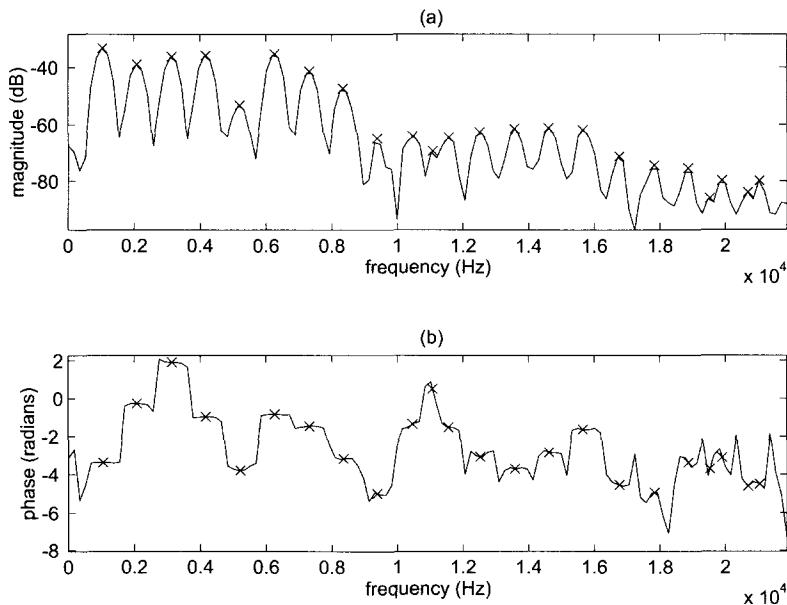


Figure 10.8 Peak detection. (a) Peaks in magnitude spectrum. (b) Peaks in the phase spectrum.

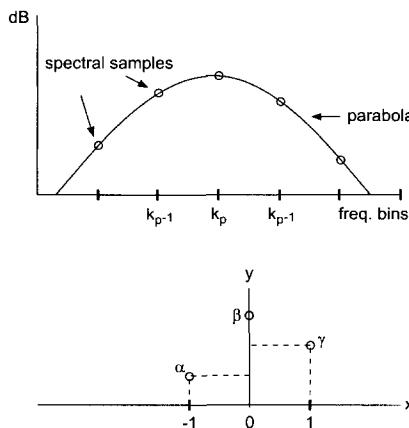


Figure 10.9 Parabolic interpolation in the peak detection process.

frequency and magnitude of a peak are obtained from the magnitude spectrum expressed in dB. Then the phase value of the peak is measured by reading the value of the unwrapped phase spectrum at the position resulting from the frequency of the peak.

Although we cannot rely on the exact shape of the peak to decide whether it is a partial or not, it is sometimes useful to have a measure of how close its shape is

to the ideal sinusoidal peak. With this idea in mind, different techniques have been used in order to improve the estimation of the spectral peaks parameters [DH97].

The M-file 10.3 implements the peak detection algorithm, the function `PickPeaks` finds the local maximums of the spectrum.

```
M-file 10.3 (PickPeaks.m)
function [loc, val] = PickPeaks(spectrum, nPeaks, minspace)
%function [loc, val] = pickpeaks(spectrum, nPeaks, minspace)
%
%==> peaking the nPeaks highest peaks in the given spectrum
%      from the greater to the lowest
% data:
%      loc: bin number of peaks  (if loc(i)==0, no peak detected)
%      val: amplitude of the given spectrum
%      spectrum: spectrum (abs(fft(signal)))
%      nPicks: number of peaks to pick
%      minspace: minimum of space between two peaks

[r, c] = size(spectrum);
rmin = min(spectrum) - 1;

% ---find a peak, zero out the data around the peak, and repeat
val = ones(nPeaks,c)*NaN;
loc = zeros(nPeaks,c);

for k=1:c      %--- find all local peaks
    difference = diff([rmin; spectrum(:,k); rmin]); % derivate
    iloc = find(difference(1:r)>= 0 & difference(2:r+1) <= 0);
                           % peak locations
    ival = spectrum(iloc,k);           % peak values

    for p=1:nPeaks
        [val(p,k),l] = max(ival);       % find current maximum
        loc(p,k) = iloc(l);           % save value and location
        ind = find(abs(iloc(l)-iloc) > minspace);
                           % find peaks which are far away
        if (isempty(ind))
            break                   % no more local peaks to pick
        end
        ival = ival(ind);           % shrink peak value and location array
        iloc = iloc(ind);
    end
end
```

The function `interpolatedValues` (M-file 10.4) computes interpolated values for each peak.

```

M-file 10.4 (interpolatedValues.m)
function [iftloc, iftphase, iftval] = interpolatedValues ...
    (r, phi, N, zp, ftloc, ftval)
%function [iftloc, iftphase, iftval] = interpolatedValues ...
%    (r, phi, N, zp, ftloc, ftval)
%
%==> computation of the interpolated values
%    of location and magnitude (parabolic interpolation)
%    and phase (linear interpolation)
%
% data:
%    iftloc:    interpolated location (bin)
%    iftval:    interpolated magnitude
%    iftphase:  interpolated phase
%    ftloc:    peak locations (bin)
%    ftval:    peak magnitudes
%    r:        magnitude of the FFT
%    phi:      phase of the FFT
%    N:        size of the FFT
%    zp:       zero-padding multiplicative coefficient

%--- calculate interpolated peak position in bins (iftloc) -----
leftftval = r((ftloc-1).*((ftloc-1)>0)+((ftloc-1)<=0).*1);
rightftval= r((ftloc+1).*((ftloc+1)<N/2)+((ftloc+1)>=N/2).*1);
leftftval = 20*log10(leftftval);
rightftval= 20*log10(rightftval);
ftval      = 20*log10(ftval);
iftloc     = ftloc + .5*(leftftval - rightftval) ./ ...
    (leftftval - 2*ftval + rightftval);

%--- interpolated ftloc -----

iftloc     = (iftloc>=1).*iftloc + (iftloc<1).*1;
iftloc     = (iftloc>N/2+1).*1 + (iftloc<=N/2+1).*iftloc;

%--- calculate interpolated phase (iphase) -----
leftftphase = phi(floor(iftloc));
rightftphase= phi(floor(iftloc)+1);
intpfactor = iftloc-ftloc;
intpfactor = (intpfactor>0).*intpfactor ...
    +(intpfactor<0).*(1+intpfactor);
diffphase   = unwrap2pi(rightftphase-leftftphase);
iftphase    = leftftphase+intpfactor.*diffphase;

%--- calculate interpolate amplitude (iftval) -----
iftval = ftval-.25*(leftftval-rightftval).*(iftloc-ftloc);

```

These functions (as well as others that will be introduced later in this chapter) make use of the `unwrap2pi` function given by M-file 10.5.

```
M-file 10.5 (unwrap2pi.m)
function argunwrap = unwrap2pi (arg)
% function argunwrap = unwrap2pi (arg)
%
%==> unwrapping of the phase, in [-pi, pi]
%   arg: phase to unwrap
arg = arg - floor(arg/2/pi)*2*pi;
argunwrap = arg - (arg>=pi)*2*pi;
```

Pitch Estimation

Although the term *pitch* should ideally be used to refer only to perceptual issues, the term *fundamental frequency* is not suitable to describe the output of techniques that will be explained herein. For that reason we will use both terms without making any distinction to refer to the output of these algorithms that aim to provide an estimation of this psychoacoustical sensation that is often (but not always) explained by the value of the fundamental frequency of a given harmonic series.

Pitch estimation is an optional step used when we know that the input sound is monophonic and pseudo-harmonic. Given this restriction and the set of spectral peaks of a frame, obtained as in the sinusoidal analysis, with magnitude and frequency values for each one, there are many possible pitch estimation strategies, none of them perfect [Hes83, MB94, Can98]. The most obvious approach is to define the pitch as the common divisor of the harmonic series that best explains the spectral peaks found in a given frame. For example, in the two-way mismatch procedure proposed by Maher and Beauchamp the estimated F_0 is chosen as to minimize discrepancies between measured peak frequencies and the harmonic frequencies generated by trial values of F_0 . For each trial F_0 , mismatches between the harmonics generated and the measured peak frequencies are averaged over a fixed subset of the available peaks. This is a basic idea on top of which we can add features and tune all the parameters for a given family of sounds.

Many trade-offs are involved in the implementation of a fundamental frequency detection system and every application will require a clear design strategy. For example, the issue of real-time performance is a requirement with strong design implications. We can add context-specific optimizations when knowledge of the signal is available. Knowing, for instance, the frequency range of the F_0 of a particular sound helps both the accuracy and the computational cost. Then, there are sounds with specific characteristics, like in a clarinet where the even partials are softer than the odd ones. From this information, we can define a set of rules that will improve the performance of the used estimator.

In the framework of the sinusoidal plus residual analysis system, there are strong dependencies between the fundamental frequency detection step and many other analysis steps. For example, choosing an appropriate window for the Fourier analysis

will facilitate detection of the fundamental and, at the same time, getting a good fundamental frequency will assist other analysis steps, including the selection of an appropriate window. Thus, it could be designed as a recursive process.

M-file 10.6 implements an algorithm for pitch detection (note that, first, different computations are accomplished in order to decide if the region being analyzed is harmonic or not).

```
M-file 10.6 (pitchDetection.m)
function[pitchvalue,pitcherror,isHarm]=pitchDetection(r,N, ...
SR,nPeaks,iftloc,iftval)
% function[pitchvalue,pitcherror,isHarm]= ...
% pitchDetection(r,N,SR,nPeaks,iftloc,iftval)
%
%==> pitch detection function, using the Two-Way Mismatch
% algorithm (see TWM.m)
%
% data:
%     r:      FFT magnitude
%     N:      size of the FFT
%     SR:      sampling rate
%     nPeaks: number of peaks tracked
%     iftloc, iftval: location (bin) and magnitude of the peak

%--- harmonicity evaluation of the signal
highenergy = sum(r(round(5000/SR*N):N/2)); % 5000 Hz to SR/2 Hz
lowenergy = sum(r(round(50/SR*N):round(2000/SR*N)));
% 50 Hz to 2000 Hz
isHarm = max(0,(highenergy/lowenergy < 0.6));

if (isHarm==1)    %-- 2-way mismatch pitch estimation when harmonic
    npitchpeaks = min(50,nPeaks);
    [pitchvalue,pitcherror] = ...
        TWM(iftloc(1:npitchpeaks),iftval(1:npitchpeaks),N,SR);
else
    pitchvalue = 0;
    pitcherror = 0;
end;

%--- in case of too much pitch error,
%   signal supposed to be inharmonic
isHarm = min (isHarm,(pitcherror<=1.5));
```

The two-way mismatch procedure is implemented as follows:

```
M-file 10.7 (TWM.m)
function [pitch, pitcherror] = TWM (iloc, ival, N, SR)
```

```
%function [pitch, pitcherror] = TWM (iloc, ival, N, SR)
%
% => Two-way mismatch error pitch detection
%     using Beauchamp & Maher algorithm
%
% data:
%     iloc:    location (bin) of the peaks
%     ival:    magnitudes of the peaks
%     N:       number of peaks
%     SR:      sampling rate

ifreq = (iloc-1)/N*SR; % frequency in Hertz

%--- avoid zero frequency peak
[zvalue,zindex] = min(ifreq);
if (zvalue==0)
    ifreq(zindex) = 1;
    ival(zindex) = -100;
end

ival2 = ival;
[MaxMag,MaxLoc1] = max(ival2);
ival2(MaxLoc1) = -100;
[MaxMag2,MaxLoc2]= max(ival2);
ival2(MaxLoc2) = -100;
[MaxMag3,MaxLoc3]= max(ival2);

%--- pitch candidates
nCand = 10; % number of candidates
pitchc = zeros(1,3*nCand);
pitchc(1:nCand)=(ifreq(MaxLoc1)*ones(1,nCand))./((nCand ...
+1-[1:nCand]));
pitchc(nCand+1:nCand*2)=(ifreq(MaxLoc2)*ones(1,nCand))./ ((nCand ...
+1-[1:nCand]));
pitchc(nCand*2+1:nCand*3)=(ifreq(MaxLoc3)*ones(1,nCand))./((nCand ...
+1-[1:nCand]));

%pitchc=100:300;
harmonic = pitchc;

%--- predicted to measured mismatch error
ErrorPM = zeros(fliplr(size(harmonic)));
MaxNPM = min(10,length(iloc));
for i=1:MaxNPM
    difmatrixPM = harmonic' * ones(size(ifreq))';
    difmatrixPM = abs(difmatrixPM ...
```

```

        -ones(fliplr(size(harmonic)))*ifreq');
[FreqDistance,peakloc] = min(difmatrixPM,[],2);
Ponddif    = FreqDistance .* (harmonic'.^(-0.5));
PeakMag    = ival(peakloc);
MagFactor  = max(0, MaxMag - PeakMag + 20);
MagFactor  = max(0, 1.0 - MagFactor/75.0);
ErrorPM    = ErrorPM ...
            +(Ponddif+MagFactor.*((1.4*Ponddif-0.5)));
harmonic   = harmonic+pitchc;
end

%--- measured to predicted mismatch error
ErrorMP = zeros (fliplr(size(harmonic)));
MaxNMP  = min(10,length(ifreq));

for i=1:length(pitchc)
    nharm      = round(ifreq(1:MaxNMP)/pitchc(i));
    nharm      = (nharm>=1).*nharm + (nharm<1);
    FreqDistance = abs(ifreq(1:MaxNMP) - nharm*pitchc(i));
    Ponddif    = FreqDistance.* (ifreq(1:MaxNMP).^(-0.5));
    PeakMag    = ival(1:MaxNMP);
    MagFactor  = max(0,MaxMag - PeakMag + 20);
    MagFactor  = max(0,1.0 - MagFactor/75.0);
    ErrorMP(i) = sum(MagFactor.*(Ponddif ...
            +MagFactor.*((1.4*Ponddif-0.5)));
end

%--- total error
Error = (ErrorPM/MaxNMP) + (0.3*ErrorMP/MaxNMP);
[pitcherror, pitchindex] = min(Error);

pitch = pitchc(pitchindex);

```

Peak Continuation

The peak detection process returns the estimated magnitude, frequency, and phase of the prominent peaks in a given frame sorted by frequency. Once the spectral peaks of a frame have been detected, and possibly a fundamental frequency identified, a peak continuation algorithm can organize the peaks into time-varying trajectories.

The output of the sinusoidal analysis is a set of spectral peak values (frequency, magnitude and phase) organized into frequency trajectories, where each trajectory models a time-varying sinusoid (see Fig. 10.10). As will be shown later, from this information we can synthesize a sound using additive synthesis. The less restrictive the peak detection step is, the more faithful the reconstruction of the original sound will be after synthesis.

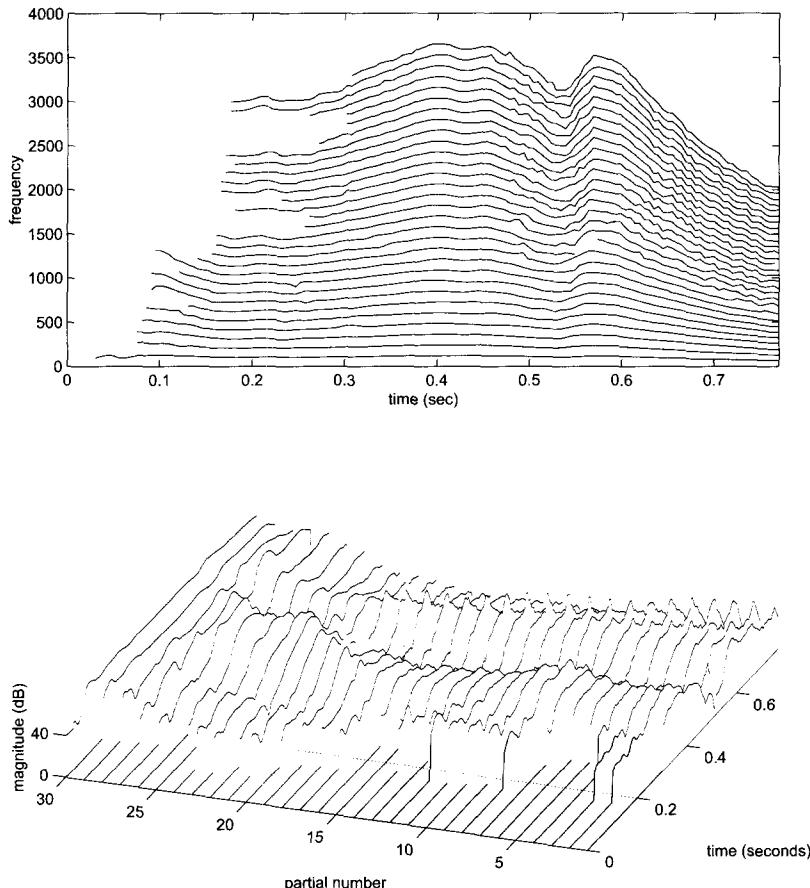


Figure 10.10 Frequency trajectories resulting from the sinusoidal analysis of a vocal sound.

The sinusoidal model assumes that each of these peaks is part of a frequency trajectory and the peak continuation algorithm is responsible for assigning each peak to a given “track”. There are many possibilities for such a process. The original one used by McAulay and Quatieri (see Fig. 10.11) in their sinusoidal representation [MQ86] is based on finding, for each peak, the closest one in frequency in the following frame.

The schemes used in the traditional sinusoidal model (as the one just mentioned), incorporate all the spectral peaks into trajectories, thus obtaining a sinusoidal representation for the whole sound. These schemes are not optimal when we want the trajectories to follow just the stable partials, leaving the rest to be modeled as part of the residual component. For example, when the partials significantly change in frequency from one frame to the next, these algorithms easily switch from the partial that they were tracking to another one, which is closer at that point.

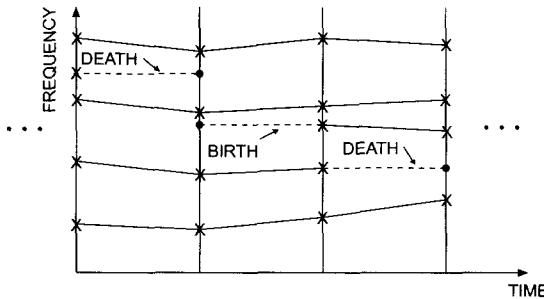


Figure 10.11 Traditional peak continuation algorithm [MQ86].

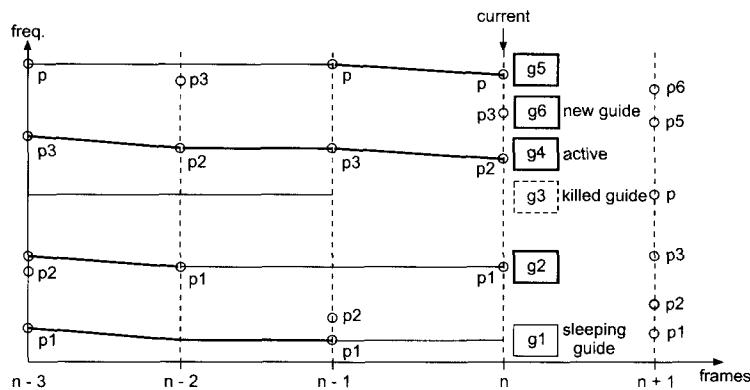


Figure 10.12 Peak continuation process. g represents the guides and p the spectral peaks.

Here we will describe a basic framework under which we can define rules for specifying the behavior of the partials of musical sounds and thus implement systems for identifying partials out of spectral peaks. The behavior of a partial, and therefore the way to track it, varies depending on the signal. Whether we have speech, a harmonic instrumental tone, a gong sound, a sound of an animal, or any other, the time-varying behavior of the partials will be different. Thus, the algorithm requires some knowledge about the characteristics of the sound that is being analyzed.

The basic idea of the algorithm is that a set of “guides” advances in time through the spectral peaks of each frame, looking for the appropriate peaks to be used (according to the specified constraints) and forming trajectories out of them (see Fig. 10.12). Thus, a guide is an abstract entity employed to create sinusoidal trajectories, which are the actual result of the peak continuation process. The instantaneous state of the guides, including their frequency and magnitude, is continuously updated as the guides are turned on, continued, and finally turned off. In the case of harmonic sounds, these guides are initialized according to the harmonic series of the detected fundamental frequency, and for inharmonic sounds, each guide is created dynamically.

The guides use the peak values and their context, such as surrounding peaks and fundamental frequency, to advance in time and form trajectories. For example, by using the detected fundamental frequency and the “memory” of an incoming trajectory to a given frame, we control the adaptation of the guides to the instantaneous changes in the sound. For a very harmonic sound, since all the harmonics evolve together, the fundamental should be the main control. Nevertheless, when the sound is less harmonic and we cannot rely on the fundamental frequency as a strong reference for all the harmonics, the information of the incoming trajectory should have a bigger weight.

Each peak is assigned to the guide that is closest to it and that is within a given frequency and amplitude deviation. If a guide does not find a match, it is assumed that the corresponding trajectory must “turn off”. In inharmonic sounds, if a guide has not found a continuation peak for a given amount of time the guide is killed. New guides, and therefore new trajectories, are created from the peaks of the current frame that are not incorporated into trajectories by the existing guides. If there are killed or unused guides, a new guide can be started. Searching through the “unclaimed” peaks of the frame for the one with the highest magnitude creates a guide. Once the trajectories have been continued for a few frames, the short ones can be deleted and we can fill the “gaps” encountered in long trajectories. A real-time implementation would not be able to use the rules that make use of the information of “future” frames.

The creation of trajectories from the spectral peaks is compatible with very different strategies and algorithms. A promising approach is to use Hidden Markov models [DGR93]. This type of approach might be very valuable for tracking partials in polyphonic sounds and complex inharmonic tones.

In our MATLAB implementation (M-file 10.8), we have chosen to provide a simple tracking algorithm that uses a simplified version of the techniques previously introduced for the case of harmonic and inharmonic sounds.

```
M-file 10.8 (peakTrackSimple.m)
function [iloc,ival,iphase,previousiloc,previousival, ...
    distminindex]=peakTrackSimple(nSines,nPeaks,N, ...
    SR,pitchvalue,iftloc,iftval,iftphase,isHarm, ...
    previousiloc,previousival);
% function [iloc,ival,iphase,previousiloc,previousival, ...
%     distminindex]=peakTrackSimple(nSines,nPeaks,N, ...
%     SR,pitchvalue,iftloc,iftval,iftphase,isHarm, ...
%     previousiloc,previousival);
%
%==> simplest partial tracking
% data:
%   iloc,ival,iphase: location (bin), magnitude
%                           and phase of peaks (current frame)
%   previousiloc,previousival,previousiphase: idem for
%                                             previous frame
%   iftloc, iftval, iftphase: idem of all of the peaks in the FT
```

```

% distminindex: indexes of the minimum distance
% between iloc and iftloc
% nPeaks: number of peaks detected
% nSines: number of peaks tracked
% N: size of the FFT
% SR: sampling rate
% pitchvalue: estimated pitch value
% isHarm: indicator of harmonicity

tmppharm = pitchvalue; %--- temporary harmonic
iloc = zeros(nSines,1);
MindB = -100;
ival = zeros(nSines,1) + MindB;
iphase = zeros(nSines,1);
distminindex = zeros(nSines,1);
Delta = 0.01;

for i=1:nSines      %--- for each sinus detected
    if (isHarm==1)      %--- for a harmonic sound
        [closestpeakmag,closestpeakindex]=min(abs((iftloc-1)/N*SR-tmppharm));
        tmppharm = tmppharm + pitchvalue;
    else                  %--- for an inharmonic sound
        [closestpeakmag,closestpeakindex]=min(abs(iftloc-previousiloc(i)));
    end
    iloc(i) = iftloc(closestpeakindex); %--- bin of the closest
    ival(i) = iftval(closestpeakindex);
    iphase(i) = iftphase(closestpeakindex);
    dist = abs(previousiloc-iloc(i));
    [distminval, distminindex(i)] = min(dist);
end

```

You will also need the code for the function `CreateNewTrack` (M-file 10.9), which implements the initialization of newborn tracks during the peak continuation process.

M-file 10.9 (CreateNewTrack.m)

```

function[newiloc,newival]=CreateNewTrack(iftloc,iftval, ...
    previousiloc,previousival,nSines,MinMag);
% function[newiloc,newival]=CreateNewTrack(iftloc,iftval, ...
%     previousiloc,previousival,nSines,MinMag);
%
%==> creation of a new track by looking for a new significant
%    peak not already tracked
% data: iftlov, iftval: bin number & magnitude of peaks detected
%         previousiloc,
%         previousival: idem for previous peaks detected

```

```
%      nSines:          number of sines
%      MinMag:          minimum magnitude (-100 dB) for
%                      0 amplitude

%--- removing peaks already tracked
for i=1:nSines
    [min, ind] = min(abs(iftval - previousival(i)));
    iftval(ind) = MinMag;
end
%--- keeping the maximum
[newival, ind] = max(iftval);
newiloc = iftloc(ind);
```

M-file 10.10 implements a visual representation of the sinusoidal tracks.

```
M-file 10.10 (PlotTracking.m)
function PlotTracking(SineFreq, pitch)
%function PlotTracking(SineFreq, pitch)
%
%==> plot the partial tracking
% data:
%      SineFreq: frequencies of the tracks
%      pitch:    frequency of the pitch
[nSines, nFrames] = size(SineFreq);

for n=1:nSines
    f=1;
    while (f<=nFrames)
        while (f<=nFrames & SineFreq(n,f)==0)
            f = f+1;
        end
        iStart = min(f,nFrames);
        while (f<=nFrames & SineFreq(n,f)>0)
            f = f+1;
        end
        iEnd = min(max(1,f-1),nFrames);
        if (iEnd > iStart)
            line((iStart:iEnd), SineFreq(n,iStart:iEnd));
        end
    end
end

h = line((1:nFrames), pitch(1:nFrames));
set(h,'linewidth', 2, 'Color', 'black');
```

Residual Analysis

Once we have identified the stable partials of a sound, we are ready to subtract them from the original signal and obtain the residual component. This subtraction can be done either in the time domain or in the frequency domain. A time domain approach requires that first the time domain signal is synthesized from the sinusoidal trajectories, while if we stay in the frequency domain, we can perform the subtraction directly in the already computed magnitude spectrum. For the time domain subtraction, the phases of the original sound have to be preserved in the synthesized signal, thus we have to use a type of additive synthesis with which we can control the instantaneous phase and this is a type of synthesis that is computationally quite expensive. On the other hand, the sinusoidal subtraction in the spectral domain is simpler but not considerably more. Our sinusoidal information from the analysis is very much undersampled, since for every sinusoid we only have the value at the top of the peaks, and thus we have to generate all the frequency samples that belong to the sinusoidal peak to be subtracted.

Once we have either the residual spectrum or the residual time signal, it is useful to study it in order to check how well the partials of the sound were subtracted and therefore analyzed. If partials remain in the residual, the possibilities for transformations will be reduced, mainly because it will not be possible to approximate the residual as a stochastic signal, thus reducing its flexibility. In this case, we should re-analyze the sound until we get a good residual, free of deterministic components. Ideally, the resulting residual should be as close as possible to a stochastic signal.

From the residual signal, we can continue our modeling strategy. To model the stochastic part of sounds, such as the attacks of most percussion instrument, the bow noise in string instruments, or the breath noise in wind instruments, we need a good time resolution and we can give up some frequency resolution. The deterministic component cannot maintain the sharpness of attacks, because, even if a high frame-rate is used, we are forced to use a long enough window, and this size determines most of the time resolution. When the deterministic subtraction is done in the time domain, the time resolution in the stochastic analysis can be improved by redefining the analysis window. The frequency domain approach implies that the subtraction is done in the spectra computed for the deterministic analysis, thus the STFT parameters cannot be changed [Ser89].

Since it is the deterministic signal that is subtracted from the original sound, measured from long windows, the resulting residual signal might have the sharp attacks smeared. To improve the stochastic analysis, we can “fix” this residual so that the sharpness of the attacks of the original sound is preserved. The resulting residual is compared with the original waveform and its amplitude re-scaled whenever the residual has a greater energy than the original waveform. Then the stochastic analysis is performed on this scaled residual. Thus, the smaller the window, the better time resolution we will get in the residual. We can also compare the synthesized deterministic signal with the original sound and whenever this signal has a greater energy than the original waveform, it means that a smearing of the deterministic component has been produced. This can be fixed somewhat by scal-

ing the amplitudes of the deterministic analysis in the corresponding frame by the difference between the original sound and the deterministic signal.

Sinusoidal Subtraction

The first step of the residual analysis is the synthesis of the sinusoidal tracks obtained as the output of the peak continuation algorithm. For a time domain subtraction (see Fig. 10.13) the synthesized signal will reproduce the instantaneous phase and amplitude of the partials of the original sound. One frame of the sinusoidal part of the sound, $d(m)$, is generated by

$$d(m) = \sum_{r=1}^R \hat{A}_r \cos[m\hat{\omega}_r + \hat{\varphi}_r], \quad m = 0, 1, 2, \dots, S - 1 \quad (10.8)$$

where R is the number of trajectories present in the current frame and S is the length of the frame. To avoid “clicks” at the frame boundaries, the parameters $\hat{A}_r, \hat{\omega}_r, \hat{\varphi}_r$ are smoothly interpolated from frame to frame.

The instantaneous amplitude $\hat{A}(m)$ is easily obtained by linear interpolation from frame to frame. Frequency and phase values are tied together (frequency is the phase derivative), and both control the instantaneous phase $\hat{\theta}(m)$, defined by

$$\hat{\theta}(m) = m\hat{\omega} + \hat{\varphi}. \quad (10.9)$$

Different approaches are possible for computing the instantaneous phase [MQ86]. Thus we are able to synthesize one frame of a sound by

$$d^l(m) = \sum_{r=1}^{R^l} \hat{A}_r^l(m) \cos[\hat{\theta}_r^l(m)] \quad (10.10)$$

which goes smoothly from the previous to the current frame with each sinusoid accounting for both the rapid phase changes (frequency) and the slowly varying phase changes.

Residual Approximation

One of the underlying assumptions of the sinusoidal plus residual model is that the residual is a stochastic signal. Such an assumption implies that the residual is fully described by its amplitude and its general frequency characteristics (see Fig. 10.14). It is unnecessary to keep either the instantaneous phase or the exact spectral shape information. Based on this, a frame of the stochastic residual can be completely characterized by the output of a filter, which has a noise input signal. The filter encodes the amplitude and general frequency characteristics of the residual. The representation of the residual for the overall sound will be a sequence of these filters, i.e., a time-varying filter.

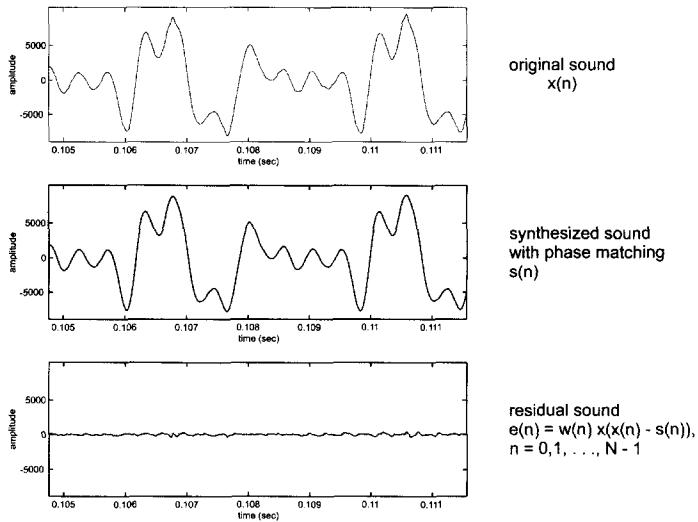


Figure 10.13 Time domain subtraction.

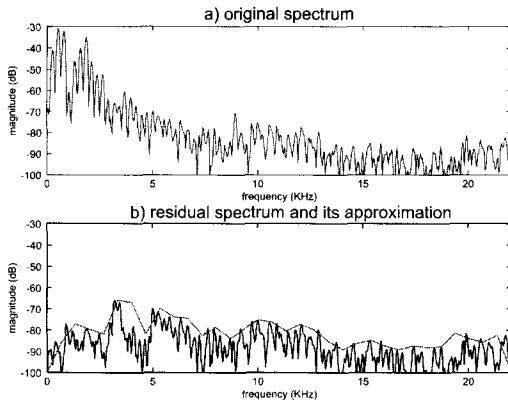


Figure 10.14 (a) Original spectrum, (b) residual spectrum and approximation.

The filter design problem is generally solved by performing some sort of curve fitting in the magnitude spectrum of the current frame [Str80, Sed88]. Standard techniques are: spline interpolation [Cox71], the method of least squares [Sed88], or straight-line approximations.

One way to carry out the line-segment approximation is to step through the magnitude spectrum and find local maxima in each of several defined sections, thus giving equally spaced points in the spectrum that are connected by straight lines to create the spectral envelope. The number of points gives the accuracy of the fit, and can be set depending on the sound complexity. Other options are unequally spaced points, for example, logarithmically spaced, or spaced according to other perceptual

criteria.

Another practical alternative, as already seen in Chapter 9 (see section 9.2.2), is to use a type of least squares approximation called linear predictive coding, LPC [Mak75, MG75]. LPC is a popular technique used in speech research for fitting an n th-order polynomial to a magnitude spectrum. For our purposes, the line-segment approach is more flexible than LPC, and although LPC results in less analysis points, the flexibility is considered more important. For a comprehensive collection of different approximation techniques of the residual component see [Goo97].

10.3.2 Feature Analysis

The accomplishment of a meaningful parameterization for sound transformation applications is a difficult task. We want a parameterization that offers an intuitive control over the sound transformation process, with which we can access most of the perceptual attributes of a sound. The analysis techniques described so far result in a simple parameterization, appropriate for describing the lower physical characteristics of the sound. In the sinusoidal plus residual model, these parameters are the instantaneous frequency, amplitude and phase of each partial and the instantaneous spectral characteristics of the residual signal.

There are other useful instantaneous attributes that give a higher-level abstraction of the sound characteristics. For example, we can describe fundamental frequency, amplitude and spectral shape of the sinusoidal component, amplitude and spectral shape of the residual component, and the overall amplitude. These attributes are calculated at each analysis frame from the output of the basic sinusoidal plus residual analysis. Afterwards, some of them can be extracted.

From a digital effects design point of view, the extraction of such attributes allows us to implement transformations that modify only one of those features without affecting the rest. A clear example is illustrated in Fig. 10.2 where the fundamental frequency is extracted, multiplied by a scaling factor, and then incorporated back into the original spectral data.

Many other features like the degree of harmonicity, noisiness, spectral tilt, or spectral centroid, can also be computed from the spectral representation of a sound. Some of them are just information attributes that describe the characteristics of the frame and have mainly found applications in sound classification tasks.

Apart from the instantaneous, or frame, values, it is also useful to have parameters that characterize the time evolution of the sound. The time changes can be described by the derivatives of each one of the instantaneous attributes.

Another important step towards a musically useful parameterization is the segmentation of a sound into regions that are homogeneous in terms of its sound attributes. Then we can identify and extract region attributes that will give higher-level control over the sound.

From the basic sinusoidal plus residual representation it is possible to extract some of the attributes mentioned above. The critical issue is how to extract them

while minimizing interferences, thus obtaining significant high level attributes free of correlations [SB98]. The general process will be to first extract instantaneous attributes and their derivatives, then segment the sound based on that information, and finally extract region attributes.

As already indicated, the basic instantaneous attributes are: amplitude of sinusoidal and residual component, overall amplitude, fundamental frequency, spectral shape of sinusoidal and residual component, harmonic distortion, noisiness, spectral centroid, and spectral tilt. These attributes are obtained at each frame using the information that results from the basic sinusoidal plus residual analysis and not taking into account the data from previous or future frames. The amplitude of the sinusoidal component is the sum of the amplitudes of all harmonics of one frame expressed in dB,

$$AS = 20\log_{10} \left(\sum_{i=1}^l a_i \right) \quad (10.11)$$

where a_i is the linear amplitude of the i th harmonic and l is the total number of harmonics found in the frame.

The amplitude of the residual component is the sum of the absolute values of the residual of one frame expressed in dB. This amplitude can also be computed by adding the frequency samples of the corresponding magnitude spectrum, according to

$$\begin{aligned} AR &= 20\log_{10} \left(\sum_{n=0}^{M-1} |x_R(n)| \right) \\ &= 20\log_{10} \left(\sum_{k=0}^{N-1} |X_R(k)| \right), \end{aligned} \quad (10.12)$$

where $x_R(n)$ is the residual sound, M is the size of the frame, $X_R(k)$ is the spectrum of the residual sound, and N is the size of the magnitude spectrum.

The total amplitude of the sound at one frame is the sum of its absolute values expressed in dB. It can also be computed by summing the amplitudes of the sinusoidal and residual components, as given by

$$\begin{aligned} A &= 20\log_{10} \left(\sum_{n=0}^{M-1} |x(n)| \right) = 20\log_{10} \left(\sum_{k=0}^{N-1} |X(k)| \right) \\ &= 20\log_{10} \left(\sum_{i=1}^l a_i + \sum_{k=0}^{N-1} |X_R(k)| \right) \end{aligned} \quad (10.13)$$

where $x(n)$ is the original sound and $X(k)$ is its spectrum.

The fundamental frequency is the frequency that best explains the harmonics of one frame. Many different algorithms can be used to compute the fundamental frequency (see previous section 10.3.1, for example) but a reasonable approximation,

once we have the sinusoidal component, can be the weighted average of all the normalized harmonic frequencies

$$F_0 = \sum_{i=1}^l \frac{f_i}{i} \cdot \frac{a_i}{\sum_{i=1}^l a_i} \quad (10.14)$$

where f_i is the frequency of the i th harmonic.

The spectral shape of the sinusoidal component is the envelope described by the amplitudes and frequencies of the harmonics, or its approximation,

$$S_{shape} = \{(f_1, a_1), (f_2, a_2), \dots, (f_l, a_l)\}. \quad (10.15)$$

The spectral shape of the residual component is an approximation of the magnitude spectrum of the residual sound of one frame. A simple function is computed as the line segment approximation of the spectrum,

$$R_{shape} = \{e_1, e_2, \dots, e_q, \dots, e_{Ncoef}\} \quad (10.16)$$

Other spectral approximation techniques can be considered depending on the type of residual and the application [Goo96].

The frame-to-frame variation of each attribute is a useful measure of its time evolution, thus an indication of changes in the sound. It is computed in the same way for each attribute,

$$\Delta = \frac{\text{Val}(l) - \text{Val}(l-1)}{H/f_s} \quad (10.17)$$

where $\text{Val}(l)$ is the attribute value for the current frame, $\text{Val}(l-1)$ is the attribute value for the previous one, H is the hop size and f_s is the sampling rate.

As an example, the following function (M-file 10.11) implements the computation of the spectral shape that will be used in some of the effects implemented in the next sections.

```
M-file 10.11 (sort.m)
%--- sorting according to the frequencies iloc
[isortedloc, ind] = sort(iloc);
isortedval = ival(ind);
[indr, indc] = find(isortedloc);
newloc = isortedloc(indr);
newval = isortedval(indr);
%--- computing the spectral shape without redundant values
spectralShape = [];
spectralShape(1,1) = 1;
spectralShape(2,1) = MinMag;
shapePos = 1;
for i=1:length(newloc)
    if newloc(i) > spectralShape(1,shapePos)
```

```

shapePos = shapePos + 1;
spectralShape(1,shapePos) = newloc(i);
spectralShape(2,shapePos) = newval(i);
end
end
%--- adding boudaries values
spectralShape(1,shapePos+1) = N/2;
spectralShape(2,shapePos+1) = MinMag;

```

Segmentation

Sound segmentation has proven important in automatic speech recognition and music transcription algorithms. For our purposes it is very valuable as a way to apply region-dependent transformations. For example, a time stretching algorithm would be able to transform the steady state regions, leaving the rest unmodified.

A musically meaningful segmentation process divides a melody into notes and silences and then each note into an attack, a steady state and a release region.

The techniques originally developed for speech [Vid90], such as those based on pattern recognition or knowledge-based methodologies, start to be used in music segmentation applications [RRSC98]. Most of the approaches apply classification methods that start from sound features, such as the ones described in this chapter, and are able to group sequences of frames into predefined categories. No reliable and general-purpose technique has been found. Our experience is that they require narrowing the problem to a specific type of musical signal or including a user intervention stage to help direct the segmentation process.

Region Attributes

Once a given sound has been segmented into regions we can compute the attributes that describe each one. Most of the interesting attributes are the mean and variance of each of the frame attributes for the whole region. For example, we can compute the spectral shape or the mean and variance for the amplitude of sinusoidal and residual components, the fundamental frequency, or the spectral tilt.

Global attributes that can characterize attacks and releases make use of the average variation of each of the instantaneous attributes, such as average fundamental frequency variation, average amplitude variation, or average spectral shape change. In the steady state regions it is important to extract the average value of each of the instantaneous attributes and measure other global attributes such as time-varying rate and depth of vibrato. Vibrato is a specific attribute present in many steady state regions of sustained instrumental sounds that requires a special treatment [HB98].

Some region attributes can be extracted from the frame attributes in the same way that they were extracted from the sinusoidal plus residual data. The result of the extraction of the frame and region attributes is a hierarchical multi-level data structure where each level represents a different sound abstraction.

10.3.3 Synthesis

From the output of the analysis techniques presented we can synthesize a new sound. The similarity with respect to the original sound will depend on how well the input sound fits the implicit model of the analysis technique and the settings of the different variables that the given technique has. In the context of the chapter we are interested in transforming the analysis output in order to produce a specified effect in the synthesized sound.

All these transformations can be done in the frequency domain. Afterwards, the output sound can be synthesized using the techniques presented in this section. The sinusoidal component will be generated using some type of additive synthesis approach and the residual, if present, will be synthesized using some type of subtractive synthesis approach.

Thus, the transformation and synthesis of a sound are done in the frequency domain; generating sinusoids, noise, or arbitrary spectral components, and adding them all to a spectral frame. Then, we compute a single IFFT for each frame, which can yield efficient implementations.

Figure 10.15 shows a block diagram of the final part of the synthesis process. Previous to that we have to transform and add all the high-level features, if they have been extracted, and obtain the lower level data (sine and residual) for the frame to be synthesized. Since the stored data might have a different frame rate, or a variable one, we also have to generate the appropriate frame by interpolating the stored ones. These techniques are presented in the following sections.

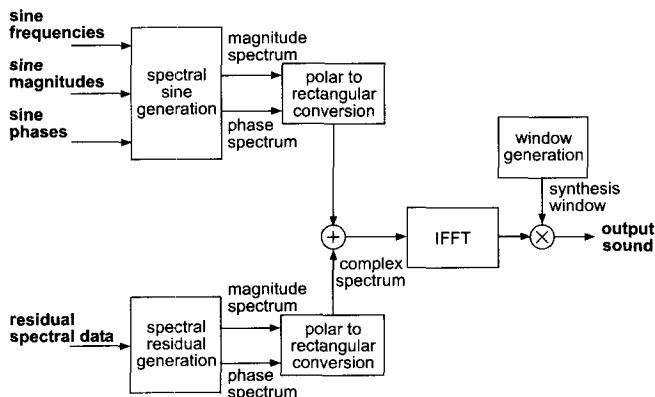


Figure 10.15 Diagram of the spectral synthesis.

Sinusoidal Synthesis

The sinusoidal component is generated with additive synthesis, similar to the sinusoidal synthesis that was part of the analysis, with the difference that now the phase trajectories might be discarded.

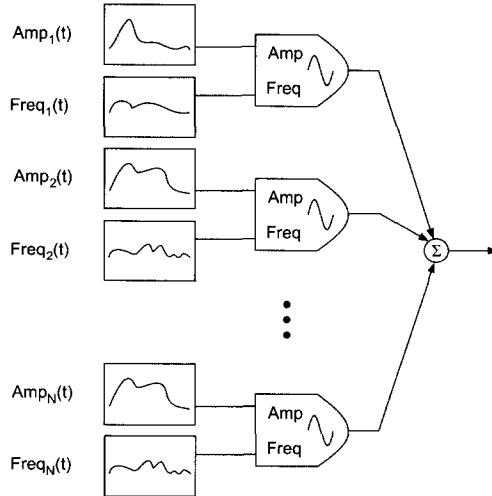


Figure 10.16 Additive synthesis block diagram.

Additive synthesis is based on the control of the instantaneous frequency and amplitude of a bank of oscillators, as shown in Fig. 10.16. The instantaneous amplitude $\hat{A}(m)$ of an oscillator is obtained by linear interpolation

$$\hat{A}(m) = \hat{A}^{l-1} + \frac{(\hat{A}^l - \hat{A}^{l-1})}{S}m, \quad (10.18)$$

where $m = 0, 1, \dots, S - 1$ is the time sample in the l^{th} synthesis frame.

The instantaneous phase is taken to be the integral of the instantaneous frequency, where the instantaneous radian frequency $\hat{\omega}(m)$ is obtained by linear interpolation

$$\hat{\omega}(m) = \hat{\omega}^{l-1} + \frac{(\hat{\omega}^l - \hat{\omega}^{l-1})}{S}m \quad (10.19)$$

and the instantaneous phase for the r^{th} sinusoid is given by

$$\hat{\theta}_r(m) = \hat{\theta}_r(l - 1) + \hat{\omega}_r(m). \quad (10.20)$$

Finally, the synthesis equation becomes

$$d^l(m) = \sum_{r=1}^{R^l} \hat{A}_r^l(m) \cos[\hat{\theta}_r^l(m)] \quad (10.21)$$

where $\hat{A}(m)$ and $\hat{\theta}(m)$ are the calculated instantaneous amplitude and phase.

A very efficient implementation of additive synthesis, when the instantaneous phase is not preserved, is based on the inverse FFT [RD92]. While this approach

loses some of the flexibility of the traditional oscillator bank implementation, especially the instantaneous control of frequency and magnitude, the gain in speed is significant. This gain is based on the fact that a sinusoid in the frequency domain is a sinc-type function, the transform of the window used, and in these functions not all the samples carry the same weight. To generate a sinusoid in the spectral domain it is sufficient to calculate the samples of the main lobe of the window transform, with the appropriate magnitude, frequency and phase values. We can then synthesize as many sinusoids as we want by adding these main lobes in the FFT buffer and performing an IFFT to obtain the resulting time-domain signal. By an overlap-add process we then get the time-varying characteristics of the sound. In M-file 10.12 we implement a sinusoidal synthesis algorithm based on this latter approach.

```
M-file 10.12 (sinefillspectrum.m)
function padsynthft=sinefillspectrum(iloc,ival,iphase,nSines, ...
    w1Length, zp, bh92SINE2SINE, bh92SINE2SINESize)
%function padsynthf =sinefillspectrum(iloc,ival,iphase,nSines, ...
%    w1Length, zp, bh92SINE2SINE, bh92SINE2SINESize)
%
%=> compute the spectrum of all the sines in the frequency
%   domain, in order to remove it from the signal
% data:
% padsynth:
% iloc, ival, iphase: location (bin), magnitude value (dB)
%                      and phase of a peak
% nSines:              number of sines (=length of ival and iloc)
% w1Length:            size of the analysis window
% zp:                  zero-padding multiplicative coefficient
% bh92SINE2SINE:      Blackman-Harris window
% bh92SINE2SINESize:  Blackman-Harris window size

peakmag=10.^((ival/20));           % magnitude (in [0;1])
halflobe=8*zp/2-1;                % bin number of the half lobe
firstbin=floor(iloc)-halflobe;    % first bin for filling positive
                                  % frequencies
firstbin2=floor(w1Length*zp-iloc+2)-halflobe;
% idem for negative frequencies
binremainder=iloc-floor(iloc);
sinphase=sin(iphase);
cosphase=cos(iphase);
findex=1-binremainder;
bh92SINE2SINEindexes=zeros(8*zp,1);
sinepadsynthft=zeros(w1Length*zp+halflobe+halflobe+1,1);
padsynthft =zeros(w1Length*zp,1);

%--- computation of the complex value
for i=1:nSines    %--- for each sine
```

```

if (iloc(i)~=0) %--- JUST WORK WITH NON ZEROS VALUES OF iloc !!!
    % -> tracked sines
beginindex = floor(0.5 + findex(i)*512/zp)+1;
bh92SINE2SINEindexes=[beginindex:512/zp:beginindex ...
+512/zp*(8*zp-1)]';
if (bh92SINE2SINEindexes(8*zp)>bh92SINE2SINEsize)
    bh92SINE2SINEindexes(8*zp)=bh92SINE2SINEsize;
end
magsin=bh92SINE2SINE(bh92SINE2SINEindexes) ...
.*sinphase(i)*peakmag(i);
magcos=bh92SINE2SINE(bh92SINE2SINEindexes) ...
.*cospahase(i)*peakmag(i);
%--- fill positive frequency
sinepadsynthft(firstbin(i)+halflobe:firstbin(i) ...
+halflobe+8*zp-1)= ...
sinepadsynthft(firstbin(i)+halflobe:firstbin(i)+ ...
halflobe+8*zp-1)+(magcos+j*magsin);
%--- fill negative frequency
if (firstbin2(i)+halflobe <= w1Length*zp)
    sinepadsynthft(firstbin2(i)+halflobe:firstbin2(i) ...
+halflobe+8*zp-1)= ...
sinepadsynthft(firstbin2(i)+halflobe:firstbin2(i)+ ...
halflobe+8*zp-1)+(magcos-j*magsin);
end
end
end

%--- fill padsynthft
padsynthft=padsynthft+sinepadsynthft(halflobe+1:halflobe+1 ...
+w1Length*zp-1);
padsynthft(1:halflobe) = padsynthft(1:halflobe) + ...
sinepadsynthft(w1Length*zp+1:w1Length*zp+halflobe);
padsynthft(w1Length*zp-halflobe+1:w1Length*zp) = ...
padsynthft(w1Length*zp-halflobe+1:w1Length*zp) ...
+ sinepadsynthft(1:halflobe);

```

The synthesis frame rate is completely independent of the analysis one. In the implementation using the IFFT we want to have a frame rate high enough so as to preserve the temporal characteristics of the sound. As in all short-time based processes we have the problem of having to make a compromise between time and frequency resolution. The window transform should have the fewest possible significant bins since this will be the number of points required to generate each sinusoid. A good window choice is the Blackman-Harris 92dB because, as already explained in section 10.3.1, its main lobe includes most of the energy. However, the problem is that such a window does not overlap perfectly to a constant in the time domain without having to use very high overlap factors, thus very high frame rates. A solu-

tion to this problem [RD92] is to undo the effect of the window by dividing by it in the time domain and applying a triangular window before performing the overlap-add process. This will give a good time-frequency compromise. The Matlab code for generating the triangular window is given by M-file 10.13.

```
M-file 10.13 (triang.m)
function w = triang(n)
% TRIANG Triangular window.
if rem(n,2)
% It's an odd length sequence
w = 2*(1:(n+1)/2)/(n+1);
w = [w w((n-1)/2:-1:1)]';
else
% It's even
w = (2*(1:(n+1)/2)-1)/n;
w = [w w(n/2:-1:1)]';
end
```

Residual Synthesis

The synthesis of the residual component of the sound is also performed in the frequency domain (see Fig. 10.17). When the analyzed residual has not been approximated, i.e. it is represented as a magnitude and phase spectrum for each frame, as a STFT, each residual spectrum is added to the spectrum of the sinusoidal component at each frame. But when a magnitude spectral envelope has approximated the residual, an appropriate complex spectrum has to be generated.

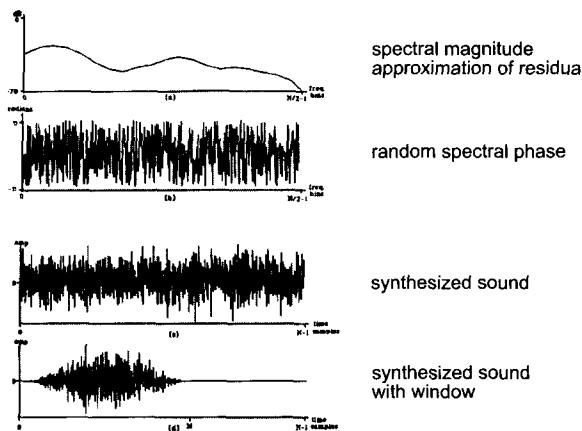


Figure 10.17 Residual synthesis approximation.

The synthesis of a stochastic signal from the residual approximation can be understood as the generation of noise that has the frequency and amplitude characteristics described by the spectral magnitude envelopes. The intuitive operation is

to filter white noise with these frequency envelopes, that is, perform a time-varying filtering of white noise, which is generally implemented using the time-domain convolution of white noise with the impulse response corresponding to the spectral envelope of a frame. We do it in the frequency domain by creating a magnitude spectrum from the approximated one, or its transformation, and generating a random phase spectrum with new values at each frame in order to avoid periodicity.

Integration of Sinusoidal and Residual Synthesis

Once the two spectral components are generated, we have to add the spectrum of the residual component to that of the sinusoids. In the process of generating the noise spectrum there has not been any window applied, since the data was added directly into the spectrum without any smoothing consideration, but in the sinusoidal synthesis we have used a Blackman-Harris 92dB, which is undone in the time domain after the IFFT. Therefore we should apply the same window to the noise spectrum before adding it to the sinusoidal spectrum. Convolving the transform of the Blackman-Harris 92dB by the noise spectrum accomplishes this, and there is only the need to use the main lobe of the window since it includes most of its energy. This is implemented quite efficiently because it only involves a few bins and the window is symmetric. Then we can use a single IFFT for the combined spectrum (see Fig. 10.18). Finally, in the time domain we undo the effect of the Blackman-Harris 92dB and impose the triangular window. By an overlap-add process we combine successive frames to get the time-varying characteristics of the sound.

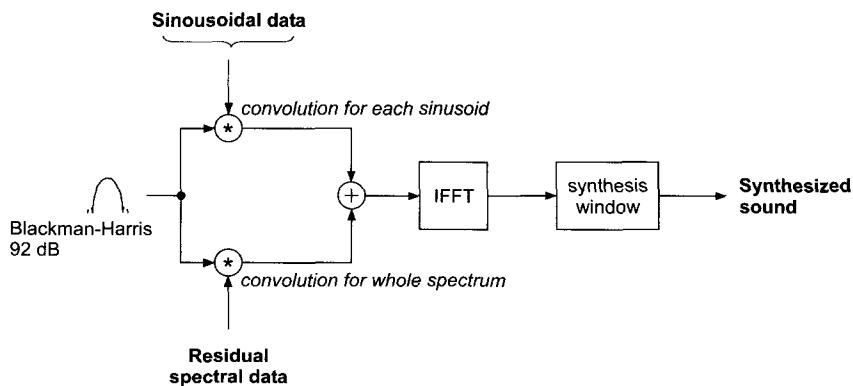


Figure 10.18 Integrating sinusoidal plus residual synthesis.

Several other approaches have been used to synthesize the output of a sinusoidal plus residual analysis. However, these techniques include modifications to the model as a whole (see [FHC00], for example).

10.3.4 Main Analysis-Synthesis Application

In M-file 10.14 we make use of all the previous functions in order to implement a complete analysis-synthesis process. We will use this framework to implement sound effects in the next sections. Note that no residual approximation is used in this implementation.

M-file 10.14 (SMS.m)

```
%=====
% SMS-Matlab like emulation
%%=====
clear all
close all
%==== USER DATA ====
DAFx_in = wavread('love.wav'); % wave file
SR = 44100; % sampling rate
w1Length = 2048; % analysis window size
n1 = 256; % analysis window hop size
nPeaks = 100; % number of peaks detected
nSines = 50; % number of sinuosoids to track (and synthetise)
minSpacePeaks = 2; % minimum space (bins) between two picked peaks
zp = 2; % zero-padding coefficient
rgain = 1.; % gain for the residual component
MaxFreq = 11000; % maximum frequency, in Hertz, for plottings
MinMag = -100; % minimum magnitude, in dB, for plottings

%--- figure data
%fig1 = 'yes'; % if uncommented, will plot the Blackman-Harris
%               % window
%fig2 = 'yes'; % if uncommented, will plot the peaks detection
%               % and tracking in one frame
%fig3 = 'yes'; % if uncommented, will plot the peak trackings
%               % real-time
%fig4 = 'yes'; % if uncommented, will plot the original and
%               % the transformed FFT in one frame
%fig5 = 'yes'; % if uncommented, will plot the peak trackings
%               % only at the end of the process
%fig6 = 'yes'; % if uncommented, will plot the original signal,
%               % its sine and residual part,
%               % and the transformed signal

%== Definition of the Windows ==
%--- definition of the analysis window
fConst=2*pi/(w1Length+1-1);
w1=[1:w1Length]';
```

```
w1=.35875 -.48829*cos(fConst*w1)+.14128*cos(fConst*2*w1) ...
-.01168*cos(fConst*3*w1);
w1=w1/sum(w1)*2;
N=w1Length*zp; % new size of the window
%--- synthesis window
w2=w1;
n2=n1;
%--- triangular window
wt2=triang(n2*2+1); % triangular window
%--- main lobe table of bh92
[bh92SINE2SINE,bh92SINE2SINESize]=bh92SINE2SINEgeneration;
%--- data for the loops
frametime = n1/SR;
pin = 0;
pout = 0;
TuneLength=length(DAFx_in);
pend=TuneLength-w1Length;

%== Definition of the data arrays ===

DAFx_in      = [zeros(w1Length/2-n1-1,1); DAFx_in];
DAFx_outsine = zeros(TuneLength,1);
DAFx_outres  = zeros(TuneLength,1);
%--- arrays for the partial tracking
iloc          = zeros(nSines,1);
ival          = zeros(nSines,1);
iphase        = zeros(nSines,1);
previousiloc  = zeros(nSines,1);
previousival  = zeros(nSines,1);
maxSines = 400; % maximum voices for harmonizer
syniloc      = zeros(maxSines,1);
synival      = zeros(maxSines,1);
previoussyniloc = zeros(maxSines,1);
previousiphase = zeros(maxSines,1);
currentiphase = zeros(maxSines,1);
%--- arrays for the sinus' frequencies and amplitudes
SineFreq = zeros(nSines,ceil(TuneLength/n2));
SineAmp  = zeros(nSines,ceil(TuneLength/n2));
pitch = zeros(1,1+ceil(pend/n1));
pitcherr = zeros(1,1+ceil(pend/n1));

%--- creating figures ---
if(exist('fig1'))
    h = figure(1); set(h,'position', [10, 45, 200, 200]);
end
if(exist('fig2'))
```

```
h = figure(2); set(h,'position', [10, 320, 450, 350]);
axisFig2 = [0 MaxFreq MinMag 0]; zoom on;
end
if(exist('fig3'))
    h = figure(3); set(h,'position', [220, 45, 550, 200]);
    axisFig3 = [1 1+ceil(pend/n1) 0 MaxFreq]; zoom on;
end
if(exist('fig4'))
    h = figure(4); set(h,'position', [470, 320, 450, 350]);
    axisFig4 = [0 MaxFreq MinMag 0]; zoom on;
end
if(exist('fig5'))
    h = figure(5); set(h,'position', [220, 45, 550, 200]);
    axisFig5 = [1 1+ceil(pend/n1) 0 MaxFreq]; zoom on;
end

%--- plot the Blackman-Harris window
if(exist('fig1'))
figure(1)
plot(20*log10(abs(fftshift(fft(bh92SINE2SINE)/bh92SINE2SINEsize))))
title('Blackman-Harris window'); xlabel('Samples');
ylabel('Amplitude')
end

tic
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
disp('analyzing frame ...');

while pin<pend
%--- windowing
grain = DAFx_in(pin+1:pin+w1Length).*w1(1:w1Length);
%--- zero padding
padgrain = zeros(N,1);
padgrain(1:w1Length/2)      = grain(w1Length/2+1:w1Length);
padgrain(N-w1Length/2+1:N) = grain(1:w1Length/2);
%--- fft computation
f    = fft(padgrain);
r    = abs(f);
phi = angle(f);
ft   = r.*exp(j*phi);

===== Analysis =====

%--- peak detection (and their plottings)
[ftloc, ftval]=PickPeaks(r(1:N/2),nPeaks,minSpacePeaks);
```

```

%--- calculate interpolated values (peak position,phase,amplitude)
[iftloc, iftphase, iftval] = ...
    interpolatedValues (r,phi,N,zp,ftloc,ftval);

%--- pitch detection
[pitchvalue,pitcherror,isHarm] = ...
    pitchDetection (r,N,SR,nPeaks,iftloc,iftval);
pitch(1+pin/n1) = pitchvalue*isHarm;
pitcherr(1+pin/n1) = pitcherror;

%--- peaks tracking
if (pin==0) %--- for the first frame
    nNewPeaks = nSines;
else %--- creating new born tracks
for i=1:nSines
    if (previousiloc(i)==0)
        [previousiloc(i), previousival(i)] = CreateNewTrack ...
            (iftloc, iftval, previousiloc, previousival, nSines, MinMag);
    nNewPeaks = nNewPeaks - 1;
    end
end

%--- simple Peak tracker
[iloc,ival,iphase,previousiloc,previousival,distminindex] = ...
    peakTrackSimple(nSines,nPeaks,N,SR,pitchvalue,iftloc, ...
        iftval,iftphase,isHarm,previousiloc,previousival);
end

%--- savings
previousival = ival;
previousiloc = iloc;
SineFreq(:,1+pin/n1)=max((iloc-1)/N*SR,0.);
    % frequency of the partials
SineAmp(:,1+pin/n1)=max(ival, MinMag);
    % amplitudes of the partials

syniloc(1:nSines) = max(1,iloc);
synival(1:nSines) = ival;

if(exist('fig3')) % plot: the trackings of partials
    figure(3); clf; hold on
    PlotTracking(SineFreq(:,1:1+pin/n1), pitch(1:1+pin/n1));
    xlabel('Frame number'); ylabel('Frequency (Hz)');
    axis(axisFig3); title('Peak tracking'); drawnow
end

```

```

%--- residual computation
resfft = ft;
if(isHarm==1)
resfft=resfft-sinefillspectrum(iloc,ival,iphase,nSines, ...
    w1Length, zp, bh92SINE2SINE, bh92SINE2SINESize);
end

%--- figures
if(exist('fig2'))
figure(2); clf; hold on
    % plot: FFT of the windowed signal (Hz,dB)
plot((1:N/2)/N*SR, 20*log10(r(1:N/2)));
for l=1:nPeaks      % plot: the peaks detected
plot([ftloc(l)-1 ftloc(l)-1]/N*SR, ...
[20*log10(ftval(l)),MinMag-1],'r:x');
end
for l=1:nSines      % plot: sines tracked and the residual part
plot([iloc(l)-1, iloc(l)-1]/N*SR, [ival(l), MinMag-1], 'k')
end
plot((1:N/2)/N*SR, 20*log10(abs(resfft(1:N/2))), 'g');
if(isHarm)          % plot: true pitch of each harmonic
for l=1:nSines
    plot([pitchvalue*l, pitchvalue*l],[1, MinMag-1], 'y:')
end
end
xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)'); axis(axisFig2);
title('Peak detection and tracking for one frame'); drawnow
end

nSynSines = nSines;
%%%%% Transformations =====
%%%%% Synthesis =====

%%% Phase computation
if (pin > 0)
    for i=1:nSynSines
        if (syniloc(i)~=0)
            ifreq = (previousssyniloc(distminindex(i))+ syniloc(i))/2;
            % average bin
            freq = (ifreq-1)/N*SR; % freq in Hz (if loc=1 --> freq=0)
            currentiphase(i)=unwrap2pi(previousiphase(distminindex(i))+...

```

```

    2*pi*freq*frametime);
end
end
end

previousssynival = synival;
previousssyniloc = syniloc;
previousiphase = currentiphase;

%--- compute sine spectrum
padsynthft=sinefillspectrum(syniloc, synival, currentiphase, ...
    nSynSines,w1Length, zp, bh92SINE2SINE, bh92SINE2SINESize);
if (isHarm==0)
padsynthft = zeros(size(padsynthft));
end

%--- residual computation
respadgrain=real(ifft(resfft));
resgrain=[respadgrain(N-w1Length/2+1:N); ...
    respadgrain(1:w1Length/2)]./w2(1:w1Length);
ressynthgrain=wt2(1:n2*2).*resgrain(w1Length/2-n2:w1Length/2+n2-1);
DAFx_outres(pout+1:pout+n2*2)=DAFx_outres(pout+1:pout+n2*2)+ ...
    ressynthgrain;

%--- sinusoidal computation
sinpadgrain=real(ifft(padsynthft));
singrain=[sinpadgrain(N-w1Length/2+1:N); ...
    sinpadgrain(1:w1Length/2)]./w2(1:w1Length);
sinsynthgrain=wt2(1:n2*2).*singrain(w1Length/2-n2:w1Length/2+n2-1);
DAFx_outsine(pout+1:pout+n2*2)=DAFx_outsine(pout+1:pout+n2*2)+ ...
    sinsynthgrain;

%--- figure with original signal and transformed signal FFT
synthr = abs(fft(respadgrain + sinpadgrain));
if(exist('fig4'))
    figure(4); clf; hold on
    plot((1:N/2)/N*SR, 20*log10(r(1:N/2)), 'b:'); axis(axisFig4);
    plot((1:N/2)/N*SR, 20*log10(synthr(1:N/2)), 'r');
    figure(4);
    xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)'); axis(axisFig4);
    title('FFT of original (blue) and transformed (red) signals');
drawnow
end

%--- increment loop indexes
pin = pin + n1;

```

```

pout = pout + n2;
disp(pin/n1);
end
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
toc

%===== write output sounds =====
DAFx_in = DAFx_in(w1Length/2-n1:length(DAFx_in));
% remove the zeros added for the process
DAFx_outresynth = DAFx_outsine(1:TuneLength)+ ...
rgain*DAFx_outres(1:TuneLength);
mm = max(abs(DAFx_outresynth));
wavwrite(DAFx_outresynth/mm, SR, 'DAFx_out.wav');
wavwrite(DAFx_outsine/mm, SR, 'DAFx_outsine.wav');
wavwrite(DAFx_outres/mm, SR, 'DAFx_outres.wav');

if(exist('fig3')==0 & exist('fig5'))% plot: trackings of partials
    % only at the end of the process
figure(5); clf; hold on
PlotTracking(SineFreq(:,1:1+pend/n1), pitch(1:1+pend/n1));
xlabel('Frame number'); ylabel('Frequency (Hz)'); axis(axisFig5);
title('Peak tracking'); drawnow
end

if(exist('fig6')) % plot the input signal, its sinus
    % and its residual part, and the transformed signal
figure(6)
subplot(4,1,1); plot(DAFx_in); xlabel('input signal');
subplot(4,1,2); plot(DAFx_outsine); xlabel('sinus part');
subplot(4,1,3); plot(DAFx_outres); xlabel('residual part');
subplot(4,1,4); plot(DAFx_outresynth);
xlabel('resynthesized signal');
end

```

10.4 FX and Transformations

In this section we introduce a set of effects and transformations based on the analysis-synthesis framework introduced throughout this chapter. All of them are accompanied by their corresponding Matlab code. In order to use them, you just have to add the code of the effect to use under the “= Transformation =” line in the main analysis-synthesis application code of the previous section.

10.4.1 Filtering with Arbitrary Resolution

Filters are probably the paradigm of a “classical” effect. Many different implementations are provided in the general DSP literature and in the previous chapters of this book. Here we introduce a different approach that differs in many aspects from the classical one.

For our “filter” implementation, we take advantage of the sinusoidal plus residual model in order to modify the amplitude of any arbitrary partial present in the sinusoidal component.

For example, we can implement a bandpass filter defined by (x, y) points where x is the frequency value in Hertz and y is the amplitude factor to apply (see Fig. 10.19). In the example code given below, we define a bandpass filter with passband range [2100 3000].

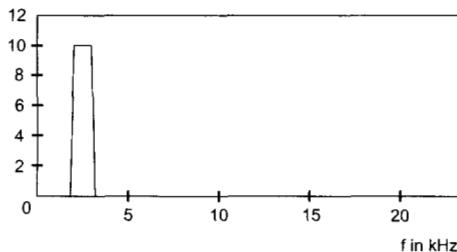


Figure 10.19 Bandpass filter with arbitrary resolution.

M-file 10.15 (ch10_t_filter_arb.m)

```
%===== Filtering with arbitrary resolution =====
Filter      = [ 0 2099 2100 3000 3001 22050 ; 0 0 1 1 0 0 ];
[syniloc,ind] = sort(iloc);
FilterEnvelope = interp1(Filter(1,:)',Filter(2,:)',syniloc/N*SR);
synival       = ival(ind)+(20*log10(max(FilterEnvelope,10^-9)));
synival(ind)  = synival;
syniloc(ind)  = syniloc;
```

As shown, our filter does not need to be characterized by a traditional transfer function, and more complex functions can be defined by summing delta-functions.

For example, the following code filters out the even partials of the input sound. If applied to a sound with a broadband spectrum, like a vocal sound, it will convert it to a clarinet-like sound.

M-file 10.16 (ch10_t_voice2clar.m)

```
%== voice to clarinet ==
syniloc = iloc;
synival = ival;
if (isHarm == 1)
```

```

for i=1:nSines
    harmNum = round(((iloc(i)-1)/w1Length*SR/2)/pitchvalue);
    if (mod(harmNum,2)==0) % case of an even harmonic number
        synival(i) = MinMag;
    end
end
end

```

10.4.2 Partial Dependent Frequency Scaling

In a similar way, we can apply a frequency scaling to the sinusoidal components of our modeled sound. In that way, we can transpose all the partials in the spectrum or reproduce pseudo-inharmonicities like frequency stretching of higher partials, which is representative of a piano sound.

In this first example we introduce a frequency shift factor to all the partials of our sound (see Fig. 10.20). Note, though, that if a constant is added to every partial of a harmonic spectrum, the resulting sound will be inharmonic.

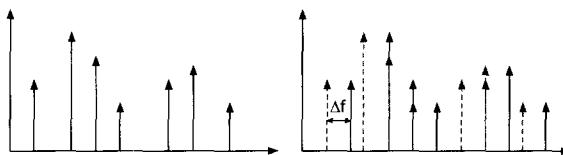


Figure 10.20 Frequency shift of the partials.

M-file 10.17 (ch10_t_freqshift.m)

```
%==== Frequency Shift =====
```

```

fstretch = 300; % frequency shift in Hz
syniloc  = iloc + round(fstretch/SR*N);
syniloc  = syniloc.*(syniloc<=N/2);

```

Another effect we can implement following this same idea is to add a stretching factor to the frequency of every partial. The relative shift of every partial will depend on its original partial index, following the formula

$$f_i = f_i \cdot f_{stretch}^{(i-1)} \quad (10.22)$$

Figure 10.21 illustrates this frequency stretching.

M-file 10.18 (ch10_t_freqstretch.m)

```
%==== Frequency Stretch =====
```

```

fstretch = 1.1;
[syniloc,ind] = sort(iloc);
syniloc = syniloc.*((fstretch).^(0:nSines-1)');
syniloc = syniloc.*(syniloc<=N/2);

```

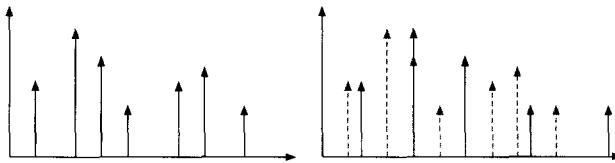


Figure 10.21 Frequency stretching.

In the same way, we can scale all the partials multiplying them by a given scaling factor. Note that this effect will act as a pitch shifter without timbre preservation.

```
M-file 10.19 (ch10_t_freqscale.m)
%%%% Frequency Scale %%%
fscale = 1.6; % frequency scaling factor
syniloc = iloc * fscale;
syniloc = syniloc.*(syniloc<=N/2);
```

10.4.3 Pitch Transposition with Timbre Preservation

In section 9.3.4, a technique was introduced in order to transpose the pitch of a sound without affecting its timbre. Here we use a similar technique in order to preserve the spectral shape of only the sinusoidal component. For that reason we scale the frequency of each partial applying the original spectral shape.

```
M-file 10.20 (ch10_t_pitchtimbre.m)
===== Pitch transposition with timbre preservation =====
if (isHarm == 1)
    pt = 2.; % pitch transposition factor
    [spectralShape,shapePos]=CalculateSpectralShape(iloc, ...
    ival,MinMag,N);
    [syniloc, synival]= PitchTransposition(iloc,ival, ...
    spectralShape,shapePos,pt,N);
    %-- comb filtering the residual
    CombCoef = 1;
    if (isHarm==1)
        resfft = CombFilter(resfft, N, SR/(pitchvalue*pt), CombCoef);
    end
end
```

The function PitchTransposition is given by:

```
M-file 10.21 (PitchTransposition.m)
===== Pitch Transposition =====
function [syniloc,synival]=PitchTransposition(iloc,ival, ...
    spectralShape,shapePos,pt,N)
syniloc = iloc.*pt;
```

```

syniloc = syniloc.*(syniloc<=N/2);
%lin. interpol. of the spectral shape for synival computation
if shapePos > 1
synival=interp1(spectralShape(1,:),spectralShape(2,:)',syniloc);
else
synival = ival;
end

```

The function CombFilter is implemented as:

```

M-file 10.22 (CombFilter.m)
function combFT = CombFilter(FT, N, delay, ampl)
%==> Comb filter in the frequency domain
% data:
%     combFT: FT of the signal comb filtered
%     FT:      FT of the signal to filtered
%     N:       size of the FT
%     delay:   delay to apply, in samples
%     ampl:    amplitude of the multiplying coefficient (in [0,1])
coef = ampl * exp(-2*j*pi*delay*(0:N-1)/N)';
combFT = FT .* (1 + coef + coef.^2);

```

Pitch Discretization to Temperate Scale

An interesting effect can be accomplished by forcing the pitch to take the nearest frequency value of the temperate scale. It is indeed a very particular case of pitch transposition where the pitch is quantified to one of the 12 semitones of an octave. This effect is widely used on vocal sounds for *dance* music and is many times referred to with the misleading name of *vocoder effect*.

```

M-file 10.23 (ch10_t_PitchDiscrete.m)
%==== Pitch discretization to temperate scale =====
if (pitchvalue ~= 0)
nst = round(12*log(pitchvalue/55)/log(2));
discpitch = 55*((2^(1/12))^nst); % discretized pitch
pt = discpitch/pitchvalue ;        % pitch transposition factor
[spectralShape,shapePos]=CalculateSpectralShape(iloc,ival, ...
MinMag,N);
[syniloc, synival]=PitchTransposition(iloc,ival,spectralShape, ...
shapePos,pt,N);
%--- comb filtering the residual
CombCoef = 1;
if (isHarm==1)
resfft = CombFilter(resfft, N, SR/(pitchvalue*pt), CombCoef);
end
end;

```

10.4.4 Vibrato and Tremolo

Vibrato and tremolo are common effects used in different kinds of acoustical instruments, including the human voice. Both are low frequency modulations: vibrato is applied to the frequency and tremolo to the amplitude of the partials. Note, though, that in this particular implementation, both effects share the same modulation frequency.

```
M-file 10.24 (ch10_t_vibtrem.m)
%%%%= vibrato and tremolo =====
if (isHarm == 1)
vtf = 5;          % vibrato-tremolo frequency in Hz
va = 10;          % vibrato depth in percentil
td = 3;           % tremolo depth in dB
synival = ival + td*sin(2*pi*vtf*pin/SR);    % tremolo
pt=1+va/200*sin(2*pi*vtf*pin/SR);% pitch transposition factor
[spectralShape, shapePos] = CalculateSpectralShape(iloc, ...
ival,MinMag,N);
[syniloc,synival]=PitchTransposition(iloc,ival,spectralShape, ...
shapePos,pt,N);
    %-- comb filtering the residual
    CombCoef = 1;
    resfft = CombFilter(resfft, N, SR/(pitchvalue*pt), CombCoef);
end
```

10.4.5 Spectral Shape Shift

As already seen in the previous chapter, many interesting effects can be accomplished by shifting the spectral shape or spectral envelope of the sinusoidal components of a sound. This shift is performed in such a way that no new partials are generated, just the amplitude envelope of the sinusoidal components is modified (see Fig. 10.22). In the following code we implement a shift of the spectral envelope by just modifying the amplitude of the partials according to the values of the shifted version of the spectral shape.

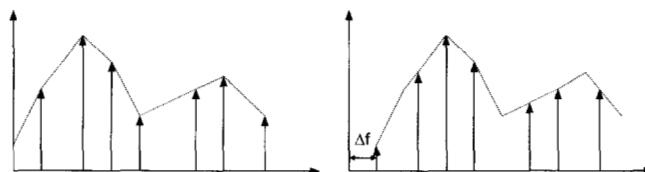


Figure 10.22 Spectral shape shift of value Δf .

M-file 10.25 (ch10_t_SpectSS.m)

```
%===== Spectral Shape Shift (positive or negative) =====
```

```

sss = -200; % spectral shape shift value in Hz
%--- spectral shape computation
[spectralShape,shapePos]=CalculateSpectralShape(iloc, ...
ival,MinMag,N);
%--- spectral shape shift
syniloc = zeros(nSines,1);
if shapePos > 1
[shiftedSpectralShape,shapePos]=SpectralShapeShift(sss, ...
iloc, ival, spectralShape, shapePos, N, SR);
end
syniloc = iloc;
%linear interpol. of the spectral shape for synival computation
if shapePos > 1
synival = interp1(shiftedSpectralShape(1,1:shapePos+1)', ...
shiftedSpectralShape(2,1:shapePos+1)', syniloc, 'linear');
else
synival = ival;
end

```

The function `SpectralShapeShift` is implemented as follows:

```

M-file 10.26 (SpectralShapeShift.m)
%===== Spectral Shape Shift =====
function [shiftedSpectralShape,shapePos]=SpectralShapeShift(sss, ...
iloc, ival, spectralShape, shapePos, N, SR)
shiftedSpectralShape = spectralShape;
sssn = round (sss*N/SR);% spectral shape shift in number of bins
if sssn > 0
    shiftedSpectralShape(1,2:shapePos)=min(N/2, ...
spectralShape(1,2:shapePos) + sssn);
    for i=shapePos:-1:1
        if shiftedSpectralShape(1,i) < N/2
            shapePos = i;
            break;
        end;
    end;
else
    shiftedSpectralShape(1,2:shapePos)= ...
max(1,spectralShape(1,2:shapePos)+ sssn);
    for i=1:shapePos
        if shiftedSpectralShape(1,i) > 1
            shiftedSpectralShape(1,2:2+shapePos+1-i) = ...
shiftedSpectralShape(1,i:shapePos+1);
            shapePos = shapePos-(i-2);
            break;
        end;
    end;

```

```

    end;
end;
```

10.4.6 Gender Change

Using the results of 10.4.3 and 10.4.5 we can change the gender of a given vocal sound. Note how by combining different “basic” effects we are able to step higher in the level of abstraction and get closer to what a naive user could ask for in a sound transformation environment, such as having a gender control on a vocal processor.

In this implementation, we apply two transformations in order to convert a male voice into a female one (variable `tr='m2f'`). The first one is a pitch transposition an octave higher. The other one is a shift in the spectral shape . The theoretical explanation to this effect is that women change their formant (resonant filters) frequencies depending on the pitch. That is, when a female singer rises up the pitch, the formants move along with the fundamental frequency.

To convert a female into a male voice (variable `tr='f2m'`) we also apply a pitch transposition and a shift in the spectral shape. This shifting has to be applied in such a way that the formants of the female voice remain stable along different pitches.

```

M-file 10.27 (ch10_t_gender.m)
%===== gender change: woman to man =====
tr='m2f'; %male to female
%tr='f2m'; %female to male
if (isHarm == 1)
pitchmin=100;
pitchmax=500;
sssmax = 50;
if (pitchvalue<pitchmin)
sss = 0;
elseif (pitchvalue>pitchmax)
sss = sssmax;
else
sss = (pitchvalue-pitchmin)/((pitchmax-pitchmin)/sssmax);
end
if(tr=='f2m')
sss=-sss;
pt=0.5;
else
pt=2;
end
%--- spectral shape computation
[spectralShape,shapePos]=CalculateSpectralShape(iloc,ival,MinMag,N);
% --- spectral shape shift
syniloc = zeros(nSines,1);
```

```

if shapePos > 1
[shiftedSpectralShape,shapePos]=SpectralShapeShift(sss,iloc,... 
  ival,spectralShape,shapePos,N,SR);
end
syniloc = iloc;
%linear interpol. of the spectral shape for synival computation
if shapePos > 1
synival = interp1(shiftedSpectralShape(1,1:shapePos+1)', ...
  shiftedSpectralShape(2,1:shapePos+1)', syniloc, 'linear');
else
synival = ival;
end
%--- pitch transposition
pt = 0.5;
[syniloc, synival] = PitchTransposition(iloc,ival,spectralShape, ...
  shapePos,pt,N);
%--- comb filtering the residual
CombCoef = 1;
if (isHarm==1)
resfft = CombFilter(resfft, N, SR/(pitchvalue*pt), CombCoef);
end
end

```

10.4.7 Harmonizer

In order to create the effect of a harmonizing vocal chorus, we can add pitch-shifted versions of the original voice (with the same timbre) and force them to be in tune with the original melody.

M-file 10.28 (ch10_t_harmonizer.m)

```

%===== harmonizer =====
nVoices    = 2;
nSynSines = nSines*(1+nVoices);
[spectralShape, shapePos] = CalculateSpectralShape(... 
  syniloc(1:nSines), synival(1:nSines), MinMag, N);
synival(1:nSines) = synival(1:nSines) - 100;
pt = [1.3 1.5]; % pitch transposition factor
ac = [-1 -2]; % amplitude change factor in dB
for i=1:nVoices
  [tmpsyniloc, tmpsynival] = PitchTransposition(... 
    syniloc(1:nSines), synival(1:nSines),...
      spectralShape, shapePos, pt(i), N);
  tmpsynival = tmpsynival + ac(i);
  syniloc(nSines*i+1:nSines*(i+1)) = tmpsyniloc;
  synival(nSines*i+1:nSines*(i+1)) = tmpsynival;
  if (pin > 0)

```

```

    distminindex(nSines*i+1:nSines*(i+1))= ...
    distminindex(1:nSines)+nSines*i;
end
end

```

10.4.8 Hoarseness

Although hoarseness is sometimes thought of as a symptom of some kind of vocal disorder [Chi94], this effect has sometimes been used by singers in order to resemble the voice of famous performers (Louis Armstrong or Tom Waits, for example). In this elemental approximation, we accomplish a similar effect by just applying a gain to the residual component of our analysis.

M-file 10.29 (ch10_t_hoarse.m)

```
rgain = 2; % gain factor applied to the residual
```

10.4.9 Morphing

Morphing is a transformation with which, out of two or more elements, we can generate new ones with hybrid properties.

With different names, and using different signal processing techniques, the idea of audio morphing is well known in the computer music community [Ser94, THH95, Osa95, SCL96]. In most of these techniques, the morph is based on the interpolation of sound parameterizations resulting from analysis/synthesis techniques, such as the short-time Fourier transform (STFT), linear predictive coding (LPC) or sinusoidal models (see cross-synthesis and spectral interpolation in sections 9.3.1 and 9.3.3, respectively).

In the following Matlab code we introduce a morphing algorithm based on the interpolation of the frequency, phase, and amplitude of the sinusoidal component of two sounds. The factor `alpha` controls the amount of the first sound we will have in the resulting morph. Different controlling factors could be introduced for more flexibility. Note, that if the sounds have different durations, the sound resulting from the morphing will have the duration of the shortest one.

Next, we include the code lines that have to be inserted in the transformation part. However, the morph transformation requires two or more inputs we have not included in order to keep the code short and understandable. Therefore the code will have to include the following modifications:

1. Read two input sounds:

```
DAFx_in1 = wavread('source1.wav');
DAFx_in2 = wavread('source2.wav');
```

2. Analyze both sounds. This means every analysis code line will have to be duplicated using the variable names:

```
iloc1, iloc2, ival1, ival2,
iphase1, iphase2, syniloc1, syniloc2, synival1, synival2,
syniphase1, syniphase2, distminindex1, distminindex2,
previousiloc1, previousiloc2, previousival1, previousival2,
pitch1, pitch2, and pitcherror1, pitcherror2.
```

M-file 10.30 (ch10_t_morph.m)

```
%== Morphing ==
%--- sorting the frequencies in bins; iloc1, iloc2
[syniloc1, ind1] = sort(iloc1);
synival1 = ival1(ind1);
syniphase1 = iphase1(ind1);
distminindex1 = distminindex1(ind1);
[syniloc2, ind2] = sort(iloc2);
synival2 = ival2(ind2);
syniphase2 = iphase2(ind2);
distminindex2 = distminindex2(ind2);
%--- interpolation -----
alpha = 0.5; % interpolation factor
syniloc = alpha*syniloc1 + (1-alpha)*syniloc2;
synival = alpha*synival1 + (1-alpha)*synival2;
%--- pitch computation
isHarmsyn = isHarm1*isHarm2;
if (isHarmsyn ==1)
npitchpeaks = min(50,nPeaks);
[pitchvalue,pitcherror]=
TWM(syniloc(1:npitchpeaks),synival(1:npitchpeaks),N,SR);
else
pitchvalue = 0;
pitcherror = 0;
end
if (pin==0) %--- for the first frame
nNewPeaks = nSines;
else
%--- creation of new born tracks
for i=1:nSines
if (previousssyniloc(i)==0)
[previousssyniloc(i),previousssynival(i)]=CreateNewTrack ...
(syniloc,synival,previousssyniloc,previousssynival,nSines,MinMag);
nNewPeaks = nNewPeaks - 1;
end
end
%--- peak tracking of the peaks of the synthetized signal
[syniloc,synival,syniphase,previousssyniloc,previousssynival, ...
distminindex]=peakTrackSimple (nSines,nPeaks,N,SR,pitchvalue, ...
syniloc,synival,syniphase,isHarmsyn,previousssyniloc, ...
```

```

previoussynival);
end

```

10.5 Content-Dependent Processing

The hierarchical data structure that includes a complete description of a given sound offers many possibilities for sound transformations. Modifying several attributes at the same time and at different abstraction levels achieve, as has already been pointed out in the previous section, most musically or end-user meaningful transformations. Higher-level transformations can refer to aspects like sound character, articulation or expressive phrasing. These ideas lead to the development of front ends such as graphical interfaces or knowledge-based systems [ALS97, ALS98] that are able to deal with the complexity of this sound representation.

In this section we introduce two applications that have been developed with these ideas in mind: a singing voice conversion and a time scaling module.

10.5.1 Real-time Singing Voice Conversion

Here we present a very particular case of audio morphing. We want to morph, in real-time, two singing voice signals in such a way that we can control the resulting synthetic voice by mixing some characteristics of the two sources. Whenever this control is performed by means of modifying a reference voice signal matching its individual parameters to another, we can refer to it as voice conversion [ANSK88].

In such a context, a karaoke-type application, in which the user can sing like his/her favorite singers, was developed [Can00]. The result is basically an automatic impersonating system that allows the user to morph his/her voice attributes (such as pitch, timbre, vibrato and articulations) with the ones from a pre-recorded singer, which from now on we will refer to as *target*.

In this particular implementation, the target's performance of the complete song to be morphed is recorded and analyzed beforehand. In order to incorporate the corresponding characteristics of the target's voice to the user's voice, the system first recognizes what the user is singing (phonemes and notes), looks for the same sounds in the target performance (i.e. synchronizing the sounds), interpolates the selected voice attributes, and synthesizes the morphed output voice. All this is accomplished in real time.

Figure 10.23 shows the general block diagram of the voice impersonator system. The system relies on two main techniques that define and constrict the architecture: the SMS framework (see 10.2.2) and a Hidden Markov model-based Speech Recognizer (SR). The SMS implementation is responsible for providing a suitable parameterization of the singing voice in order to perform the morph in a flexible and musically-meaningful way. On the other hand, the SR is responsible for matching the singing voice of the user with the target.

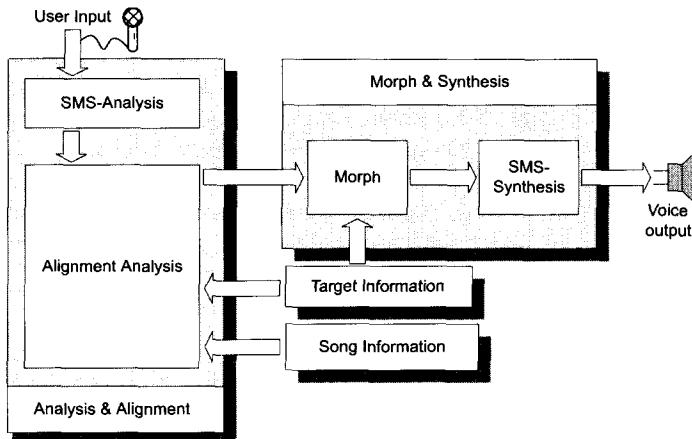


Figure 10.23 System block diagram.

Let us take an overview of the whole process. Before we can morph a particular song, we have to supply information about the song to be morphed and the song recording itself (Target Information and Song Information). The system requires the phonetic transcription of the lyrics, the melody as MIDI data, and the actual recording to be used as the target audio data. Thus, a good impersonator of the singer that originally sang the song has to be recorded. This recording has to be analyzed with SMS, segmented into morphing units (phonemes), and each unit labeled with the appropriate note and phonetic information of the song. This preparation stage is done semi-automatically, using a non-real-time application developed for this task.

Once we have all the required inputs set we can start processing the user's voice. The first module of the running system includes the real-time analysis and the recognition/alignment steps. Each analysis frame, with the appropriate parameterization, is associated with the phoneme of a specific moment of the song and thus with a target frame. Once a user frame is matched with a target frame, we morph them by interpolating data from both frames and we synthesize the output sound. Only voiced phonemes are morphed and the user has control over which parameters are interpolated, and by how much. The frames belonging to unvoiced phonemes are left untouched, thus always having the user's unvoiced consonants in the output.

Several modifications are done to the basic SMS procedures to adapt them to the requirements of the impersonator system. The major changes include the real-time implementation of the whole analysis/synthesis process with a processing latency of less than 30 milliseconds and the tuning of all parameters to the particular case of the singing voice. These modifications include the extraction of higher-level parameters meaningful in the case of the singing voice and that will be later used in the morphing process.

The system includes an Automatic Speech Recognizer (ASR) based on phoneme-base discrete Hidden Markov models in order to solve the matching problem. This ASR has been adapted to handle musical information and works with very low delay [LCB99] since we cannot wait for a phoneme to be finished before it is recognized, moreover, we have to assign a phoneme to each frame. This would be a rather impossible/impractical situation if it were not for the fact that the lyrics of the song are known beforehand. This reduces a large portion of the search problem: all the possible paths are restricted to just one string of phonemes, with several possible pronunciations. The problem is cut down to the question of locating the phoneme in the lyrics and positioning the start and end points.

As well as knowing the lyrics, musical information is also available. The user is singing along with the music, and hopefully according to a tempo and melody already specified in the score. Thus, we also know the time at which a phoneme is supposed to be sung, its approximate duration, its associated pitch, etc. All this information is used to improve the performance of the recognizer and also allows resynchronization, for example, in the case of a singer skipping a part of the song.

Depending on the phoneme the user is singing, a unit from the target is selected. Each frame from the user is morphed with a different frame from the target, advancing sequentially in time. Then the user has the choice of interpolating the different parameters extracted at the analysis stage, such as amplitude, fundamental frequency, spectral shape, residual signal, etc. In general, the amplitude will not be interpolated, thus always using the amplitude from the user and the unvoiced phonemes will not be morphed either, thus always using the consonants from the user. This will give the user the feeling of being in control. This recognition and matching process is illustrated in Fig. 10.24.

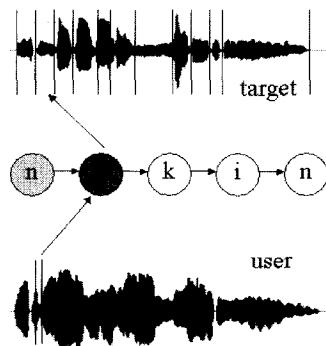


Figure 10.24 Recognition and matching of morphable units.

Whenever the spectral shape is interpolated, and the morph factor is set around 50 percent, the resulting spectral shape is smoothed and loses much of its timbre characteristic. This problem can be solved if formants are included in the spectral shape model and they are taken into account in the interpolation step.

In most cases, the durations of the user and target phonemes to be morphed will be different. If a given user's phoneme is shorter than the one from the target, the system will simply skip the remaining part of the target phoneme and go directly to the articulation portion. In the case when the user sings a longer phoneme than the one present in the target data, the system enters in the loop mode. Each voiced phoneme of the target has a loop point frame, marked in the pre-processing, non-real time stage. The system uses this frame for loop-synthesis in case the user sings beyond that point in the phoneme. Once we reach this frame in the target, the rest of the frames of the user will be interpolated with that same frame until the user ends the phoneme. This process is illustrated in Fig. 10.25.

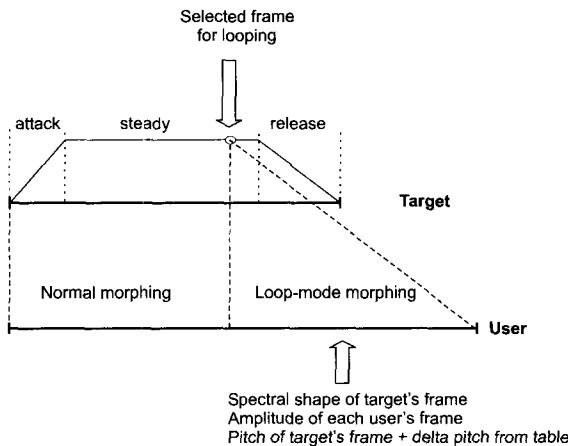


Figure 10.25 Loop synthesis diagram.

The frame used as a loop frame requires a good spectral shape and, if possible, a pitch very close to the note that corresponds to that phoneme. Since we keep a constant spectral shape, we have to do something to make the synthesis sound natural. The way we do it is by using some “natural” templates obtained from the analysis of a longer phoneme that are then used to generate more target frames to morph with the loop frame. For example, one feature that adds naturalness is pitch variations of a steady state note sung by the same target. These delta pitches are kept in a look-up table whose first access is random and consecutive values are read afterwards. Two tables are kept, one with variations of steady pitch and another one with vibrato to generate target frames.

Once all the chosen parameters have been interpolated in a given frame, they are added to the basic synthesis frame of the user. The synthesis is done with the standard synthesis procedures of SMS.

10.5.2 Time Scaling

Time scaling an audio signal means changing the length of the sound without affecting other perceptual features, such as pitch or timbre. Many different techniques,

both in time and frequency domain, have been proposed to implement this effect. Some frequency domain techniques yield high-quality results and can work with large scaling factors. However, they are bound to present some artifacts, like phasiness, loss of attack sharpness and loss of stereo image. In this section we will present a frequency domain technique for near lossless time-scale modification of a general musical stereo mix [Bon00].

The Basic System

The general block diagram of the system is represented in Fig. 10.26. First, the input sound is windowed and applied to the FFT which gives the analysis frame AF_n , that is, the spectrum bins and the amplitude and phase envelopes (n is the analysis frame index). Then the time scaling module generates the synthesis frame SF_m that is fed to the inverse FFT (IFFT, m is the synthesis frame index). Finally, the windowing and overlap-add block divides the sound segment by the analysis window and multiplies it by the overlap-add window, to reconstruct the output sound. The basics of the FFT/IFFT approach are detailed in Chapter 8.

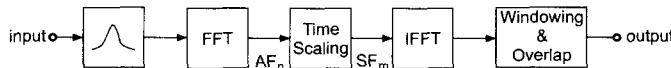


Figure 10.26 Block diagram of a general time scaling system based on the FFT/IFFT approach.

It is important to remark that the frame rate used in both the analysis and synthesis modules is the same, as opposed to the most broadly used time scaling techniques in which a change of frame rate in synthesis is used in order to achieve the effect. The window size and type must also be the same in both processes.

Figure 10.27 illustrates the operations for a time-scale stretching factor $TS > 1$, and a time compression factor $TS < 1$. The horizontal axis corresponds to the time of the center of the frame in the input audio signal. Therefore, when $TS > 1$, the time increments relative to the input audio signal will be shorter in the synthesis than in the analysis frames, but the actual frame rate will be exactly the same. Each synthesis frame points to the nearest look-ahead analysis frame. In some cases, as shown in Fig. 10.27, an analysis frame is used twice (or more) while in other cases some frames are never used. This technique will not add any artifacts, provided the frame size we use is small enough and the sound does not present abrupt changes in that particular region. In the case of a percussive attack, though, a frame repetition or omission can be noticed regardless of the analysis frame size. Therefore, some knowledge of the features of a sound segment is needed to decide where this technique can or cannot be applied.

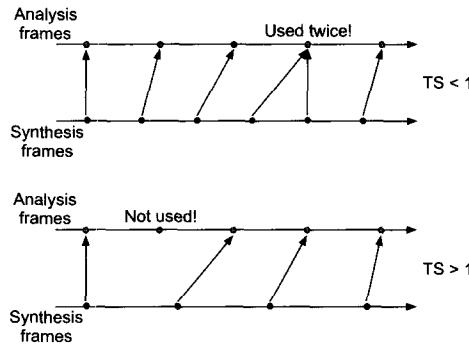


Figure 10.27 Analysis and synthesis frames.

Figure 10.28 shows a detailed block diagram of the time scaling module. The analysis frames AF_n , containing the spectrum amplitude and phase envelopes, are fed to the time scaling module. This module performs a peak detection and a peak continuation algorithm (see 10.3.1) on the current and previous z^{-1} amplitude envelopes. Then, only the peaks that belong to a sinusoidal track are used as inputs to the spectrum phase generation module. Note that the time scaling module only changes the phase, leaving the spectral amplitude envelope as it is.

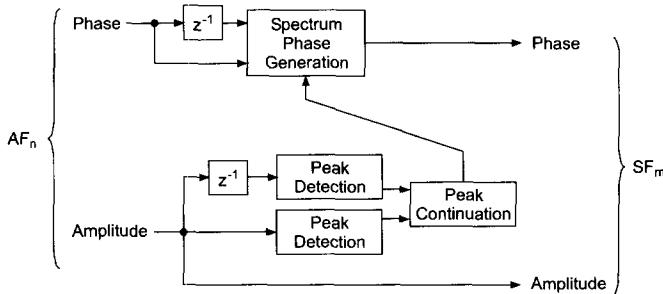


Figure 10.28 The time scaling module.

The phase of each peak is computed supposing that the frequency varies linearly between two consecutive frames and that there is some phase deviation $\Delta\varphi$ (see Fig. 10.29). The usage of the same frame rate in analysis and synthesis allows us to suppose that the phase variation between two consecutive frames is also the same.

Common Problems and Solutions in Time Scaling

Chapter 8 has already introduced a spectral technique for time scaling based on the phase vocoder approach. This kind of implementation presents very well-known artifacts. In this section we will describe some of these problems and the solution that the implementation we are proposing can provide.

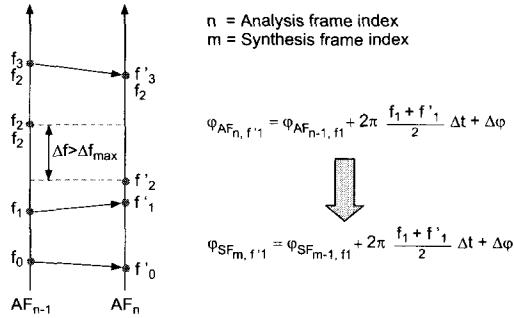


Figure 10.29 Peak continuation and phase generation.

Phasiness. In the phase vocoder implementation, the original frame has a flat phase envelope around the peak because of the circular convolution of the analysis window with the sinusoid. But after time scaling is applied, the phase loses its original behavior. This artifact is introduced due to the fact that the phase of each bin advances at different speed (see section 8.4.3). This loss of peak's phase coherence is known as *phasiness*. To avoid this problem we can apply the original relative behavior of the phase around the peak. As pointed out in [LD97], each peak subdivides the spectrum into a different region, with a phase related to that of the peak. The phase around each peak is obtained applying the delta phase function of the original spectrum phase envelope (see Fig. 10.30).

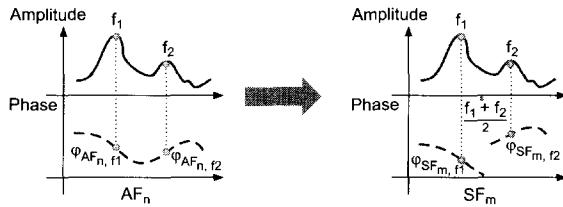


Figure 10.30 Original delta phase function around each peak.

Loss of attack transients. Another typical artifact of the phase vocoder approach is the smoothing of the attack transients. A possible solution is to modify the sinusoidal plus residual model in order to model these transients [VM98]. Another possible approach is not to time-scale the input signal on this kind of regions so that the original timing is respected (see Fig. 10.31). Consequently, and in order to preserve the overall scaling factor, a greater amount of scaling should be applied to surrounding regions.

In order to apply the previous technique, it is necessary to detect attack transients of the sound in an unsupervised manner. The computation of relative changes of energy along several frequency bands can be used for that purpose. A low frequency band could, for example, detect sharp bass notes, while a high frequency band could be set to detect hits of a crash cymbal.

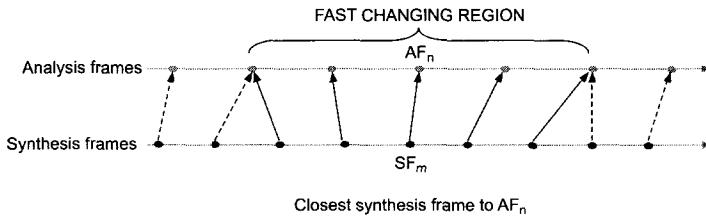


Figure 10.31 Attack transient region.

The spectrum of the input signal is given by

$$X(sR_a, k) = |X(sR_a, k)| \cdot e^{j\varphi(sR_a, k)} \quad (10.23)$$

where the FFT has been sampled every R_a samples in time, and s is the time index of the short-term transform. If we define a set of frequency bands $B_i(k)$, then the energy of the i^{th} band can be computed as

$$E(s, i) = \sum_{i=0}^{N-1} B_i(k) \cdot X^2(sR_a, k) \quad (10.24)$$

and the relative change of energy $C(s, i)$ at frame s as

$$C(s, i) = \frac{-2E(s-2, i) - E(s-1, i) + E(s+1, i) + 2E(s+2, i)}{E(s, i)}. \quad (10.25)$$

The maxima of $C(s, i)$ over some threshold should then indicate the attack transients of the input signal at the desired band.

Frequency versus time resolution. As explained in 10.3.1, it is desirable to have long windows in order to achieve a high frequency resolution, but also to have short windows so to achieve a better time resolution. If the audio signal presents an important low frequency component, the use of a long window is a must, because the low frequency peaks will be too close to be distinguishable if we use a short window. On the other hand if we apply a very long window, the time scaling process will add reverb and will smooth the sound.

The solution proposed is to use parallel windowing, that is, several analysis channels (see Fig. 10.32). Each channel is the result of an FFT with a specific window size, window type and zero-padding. Obviously, the window should be longer for low frequencies than for high frequencies. The peak detection process is applied to each of the channels while the peak continuation takes care of the desired channel frequency cuts, so it can connect peaks of different channels. Then the time scaling module fills the spectrum of all the channels (amplitude and phase envelopes) and applies a set of parallel filters $H_k(f)$ that must add up to a constant (allpass filter).

If the cut-off frequency of a channel was close to a spectral peak, this would be broken apart into two different channels and we would be introducing some kind

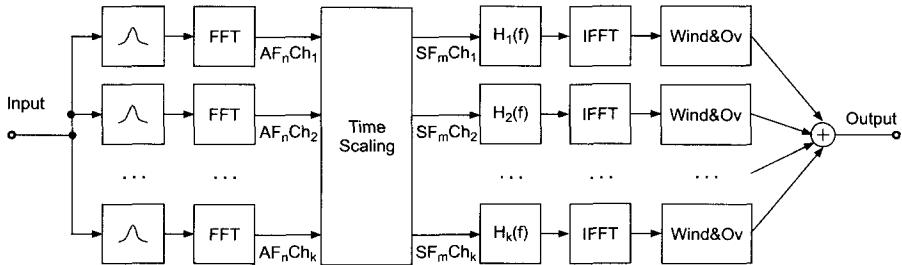


Figure 10.32 Multiple parallel windowing.

of artifacts. For that reason, and in order to guarantee that phase and amplitude envelopes around the peak behave the way we expect, we need to provide our system with time-varying frequency cuts. Each frequency cut is computed as the middle point between the two closest peaks to the original frequency cut (see Fig. 10.33).

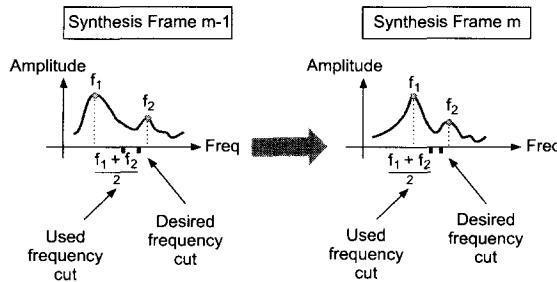


Figure 10.33 Variable phase frequency cut.

Loss of stereo image. In the case of stereo signals, if we process each one of the channels independently, most of the stereo image is bound to be lost. This artifact is mainly due to the fact that the time scaling process changes the phase relationship between the two channels. Therefore, if we want to keep the stereo image, it is necessary to preserve the phase and amplitude relationship between left and right channels.

The fact that the system does not change the amplitude envelope of the spectrum guarantees that the amplitude relationship between channels will be preserved, provided we always use frames with identical time tags for both channels. For that purpose, we need to synchronize the attack transients between the two channels.

Figure 10.34 shows the simplified block diagram of the stereo time scaling system. Notice that the number of FFT and IFFT operations is multiplied by two and, as a consequence, the same happens to the processing time.

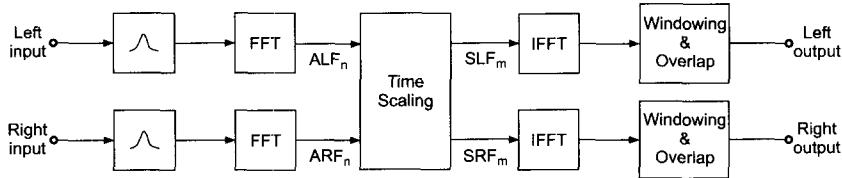


Figure 10.34 Stereo time scaling.

Time-varying Time Scaling

The system presented can deal with time-varying scaling factors with no loss of quality trade-off. The only significant change is that the time increments of the synthesis frames in the input signal are not constant.

The application of time-varying tempo variations opens up many new and interesting perspectives. The system could easily be adapted and used for alignment and synchronization of two sound sources. Also, the amount of time scaling could be used in a wise way to inspire emotions. For example, to increase the climax or the suspense of a musical piece, by slowing or increasing the tempo during certain fragments. Another interesting application could be to control the time scaling factor in the same way as the orchestra conductor does and play in real time a previously recorded background with a live performance.

10.6 Conclusion

Throughout this chapter, we have shown how the use of higher-level spectral models can lead to new and interesting sound effects and transformations. We have also seen that it is not easy nor immediate to get a good spectral representation of a sound, so the usage of this kind of approach needs to be carefully considered bearing in mind the application and the type of sounds we want to process. For example, most of the techniques presented here work well only on monophonic sounds and some rely on the pseudo-harmonicity of the input signal.

Nevertheless, the use of spectral models for musical processing has not been around too long and it has already proven useful for many applications, as the ones presented in this chapter. Under many circumstances, higher-level spectral models, such as the sinusoidal plus residual, offer much more flexibility and processing capabilities than more immediate representations of the sound signal.

In general, higher-level sound representations will offer more flexibility at the cost of a more complex and time-consuming analysis process. It is important to remember that the model of the sound we choose will surely have great effect on the kind of transformations we will be able to achieve and on the complexity and efficiency of our implementation. Hopefully, the reading of this chapter, and the book as a whole, will guide the reader towards making the right decisions in order to get the desired results.

Bibliography

- [ALS97] J.L. Arcos, R. López de Mántaras, and X. Serra. Saxex: a case-based reasoning system for generating expressive musical performances. In *Proc. International Computer Music Conference*, pp. 25–30, 1997.
- [ALS98] J.L. Arcos, R. López de Mántaras, and X. Serra. Saxex: a case-based reasoning system for generating expressive musical performances. *Journal of New Music Research*, 27(3):194–210, September 1998.
- [ANSK88] M. Abe, S. Nakamura, K. Shikano, and H. Kuwabara. Voice conversion through vector quantization. In *Proc. IEEE ICASSP-1988*, pp. 655–658, 1988.
- [Bon00] J. Bonada. Automatic technique in frequency domain for near-lossless time-scale modification of audio. In *Proc. International Computer Music Conference*, pp. 396–399, 2000.
- [Can98] P. Cano. Fundamental frequency estimation in the SMS analysis. In *Proc. DAFX-98 Digital Audio Effects Workshop*, pp. 99–102, Barcelona, November 1998.
- [Can00] P. Cano, A. Loscos, J. Bonada, M. de Boer, and X. Serra. Voice morphing system for impersonating in karaoke applications. In *Proc. International Computer Music Conference*, pp. 109–112, 2000.
- [Chi94] D.G. Childers. Measuring and modeling vocal source-tract interaction. *IEEE Transactions on Biomedical Engineering*, 41(7):663–671, July 1994.
- [Cox71] M.G. Cox. An algorithm for approximating convex functions by means of first-degree splines. *Computer Journal*, 14:272–275, 1971.
- [DGR93] Ph. Depalle, G. Garcia, and X. Rodet. Analysis of sound for additive synthesis: tracking of partials using Hidden Markov models. In *Proc. International Computer Music Conference*, pp. 94–97, 1993.
- [DH97] Ph. Depalle and T. Hélie. Extraction of spectral peak parameters using a short-time Fourier transform modeling and no sidelobe windows. In *Proceedings of the 1997 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pp. 298–231, Monhonk, 1997.
- [DQ97] Y. Ding and X. Qian. Sinusoidal and residual decomposition and residual modeling of musical tones using the QUASAR signal model. In *Proc. International Computer Music Conference*, pp. 35–42, 1997.
- [FHC00] K. Fitz, L. Haken, and P. Christensen. A new algorithm for bandwidth association in bandwidth-enhanced additive sound modeling. In *Proc. International Computer Music Conference*, pp. 384–387, 2000.

- [Goo96] M. Goodwin. Residual modeling in music analysis-synthesis. In *Proc. IEEE ICASSP-1996*, pp. 1005–1008, Atlanta, 1996.
- [Goo97] M. Goodwin. *Adaptative Signal Models: Theory, Algorithms and Audio Applications*. PhD thesis, University of California, Berkeley, 1997.
- [Har78] F.J. Harris. On the use of windows for harmonic analysis with the discrete Fourier transform. *Proceedings IEEE*, 66:51–83, 1978.
- [HB98] P. Herrera and J. Bonada. Vibrato extraction and parameterization in the spectral modeling synthesis framework. In *Proc. DAFX-98 Digital Audio Effects Workshop*, pp. 107–110, Barcelona, November 1998.
- [Hes83] W. Hess. *Pitch Determination of Speech Signals*. Springer-Verlag, 1983.
- [LCB99] A. Loscos, P. Cano, and J. Bonada. Low-delay singing voice alignment to text. In *Proc. International Computer Music Conference*, pp. 437–440, 1999.
- [LD97] J. Laroche and M. Dolson. About this phasiness business. In *Proc. International Computer Music Conference*, pp. 55–58, 1997.
- [Mak75] J. Makhoul. Linear prediction: a tutorial review. *Proceedings of the IEEE*, 63(4):561–580, 1975.
- [MB94] R.C. Maher and J.W. Beauchamp. Fundamental frequency estimation of musical signals using a two-way mismatch procedure. *J. Acoust. Soc. Am.*, 95(4):2254–2263, 1994.
- [MG75] J.D. Markel and A.H. Gray. *Linear Prediction of Speech*. Springer-Verlag, 1975.
- [MQ86] R.J. McAulay and T.F. Quatieri. Speech analysis/synthesis based on a sinusoidal representation. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 34(4):744–754, 1986.
- [Osa95] N. Osaka. Timbre interpolation of sounds using a sinusoidal model. In *Proc. International Computer Music Conference*, pp. 408–411, 1995.
- [RD92] X. Rodet and Ph. Depalle. Spectral envelopes and inverse FFT synthesis. *Proc. 93rd AES Convention*, San Francisco, AES Preprint No. 3393 (H-3), October 1992.
- [RRSC98] S. Rossignol, X. Rodet, J. Soumagne, J.-L. Collette, and Ph. Depalle. Feature extraction and temporal segmentation of acoustic signals. In *Proc. International Computer Music Conference*, 1998.
- [SB98] X. Serra and J. Bonada. Sound transformations based on the SMS high level attributes. In *Proc. DAFX-98 Digital Audio Effects Workshop*, pp. 138–142, Barcelona, November 1998.

- [SCL96] M. Slaney, M. Covell, and B. Lassiter. Automatic audio morphing. In *Proc. IEEE ICASSP-1996*, pp. 1001–1004, 1996.
- [Sed88] R. Sedgewick. *Algorithms*. Addison-Wesley, 1988.
- [Ser89] X. Serra. *A System for Sound Analysis/Transformation/Synthesis based on a Deterministic plus Stochastic Decomposition*. PhD thesis, Stanford University, 1989.
- [Ser94] X. Serra. Sound hybridization techniques based on a deterministic plus stochastic decomposition model. In *Proc. International Computer Music Conference*, pp. 348–351, 1994.
- [Ser96] X. Serra. Musical sound modeling with sinusoids plus noise. In G. De Poli, A. Picialli, S. T. Pope, and C. Roads (eds), *Musical Signal Processing*, pp. 91–122, Swets & Zeitlinger Publishers, 1996.
- [SMS] www.iua.upf.es/~sms
- [SS87] J.O. Smith and X. Serra. PARSHL: an analysis/synthesis program for non-harmonic sounds based on a sinusoidal representation. In *Proc. International Computer Music Conference*, pp. 290–297, 1987.
- [SS90] X. Serra and J. Smith. Spectral modeling synthesis: a sound analysis/synthesis system based on a deterministic plus stochastic decomposition. *Computer Music Journal*, 14(4):12–24, 1990.
- [Str80] J. Strawn. Approximation and syntactic analysis of amplitude and frequency functions for digital sound synthesis. *Computer Music Journal*, 4(3):3–24, 1980.
- [THH95] E. Tellman, L. Haken, and B. Holloway. Timbre morphing of sounds with unequal numbers of features. *J. Audio Eng. Soc.*, 43(9):678–689, 1995.
- [Vid90] E. Vidal and A. Marzial. A review and new approaches for automatic segmentation of speech signals. In L. Torres and others (eds), *Signal Processing V: Theories and Applications*, pp. 43–53, Elsevier Science Publishers, 1990.
- [VM98] T.S. Verma and T.H.Y. Meng. Time scale modification using a sines+transients+noise signal model. In *Proc. DAFX-98 Digital Audio Effects Workshop*, pp. 49–52, Barcelona, November 1998.
- [VM00] T.S. Verma and T.H.Y. Meng. Extending spectral modeling synthesis with transient modeling synthesis. *Computer Music Journal*, 24(2):47–59, 2000.

Chapter 11

Time and Frequency Warping Musical Signals

G. Evangelista

11.1 Introduction

In this chapter we describe interesting audio effects that can be obtained by deforming the time and/or the frequency axis. Whilst discrete-time warping techniques were introduced in 1965 [Bro65], their interest in musical applications is fairly recent. Time warping aims at deforming the waveform or the envelope of the signal while frequency warping modifies its spectral content, e.g., by transforming an harmonic signal into an inharmonic one or vice versa. The effects obtained by warping often increase the richness of the signal by introducing detuning or fluctuation of the waveform. The sounds from natural instruments like piano and drums already possess this property. The wave propagation in stiff strings and membranes can actually be explained in terms of frequency warping. By warping these sounds one can enhance or reduce their natural features. Even uninteresting synthetic sounds such as pulse trains may be transformed into interesting sounds by warping. Frequency warping is amenable to a time-varying version that allows us to introduce dynamic effects such as vibrato, tremolo, Flatterzunge in flute.

The quality of warping ultimately depends on the warping map, i.e., on the function describing the deformation of the time or frequency axis. Time and frequency warping are flexible techniques that give rise to a tremendous amount of possibilities, most of which are at present still unexplored from a musical point of view. By choosing the proper map one can actually morph the sound of an instrument into that produced by another instrument.

This chapter is divided into two main sections. In the first section we describe the time and frequency warping operations and derive algorithms for computing

these effects. In the second section we illustrate some of their musical applications based on examples and case studies.

11.2 Warping

11.2.1 Time Warping

Suppose that we want to change the shape of a periodic waveform $s(t)$ by moving the amplitude values attained by the signal to other time instants. One can achieve this by plotting the signal on an elastic sheet and by stretching and/or compressing the sheet in different points along its horizontal direction. The waveshape appears as if the original time axis had been deformed. Instants of time that were equidistant now have a different time distribution. This deformation of the time axis called time warping is characterized by a warping map $\theta(t)$ mapping points of the original t -axis into points of the transformed axis. An example of time warping a sinewave is shown in Fig. 11.1. The figure is obtained by plotting the original signal along the ordinates and transforming time instants into new time instants via the warping map, obtaining the signal plotted along the abscissa axis. Notice that to one point of the original signal there can correspond more points of the warped signal. These points are obtained by joining time instants of the original signal to points on the warping characteristic $\theta(t)$ using horizontal lines. The corresponding warped time instants are the value(s) of the abscissa corresponding to these intersection point(s). The time warped signal is obtained by plotting the corresponding amplitude values at the new time instants along the abscissa. In this example the signal $\sin(\theta(t))$ may be interpreted as a phase modulation of the original sinewave. Time warping a signal composed of a superposition of sinewaves is equivalent to phase modulating each of the component sinewaves and adding them together. By time warping we alter not only the waveshape but also the period of the signal. Clearly, the map is effective modulo the period of the signal, that is, the map $\theta(t)$ and the map

$$\theta_0(t) = \text{rem}(\theta(t), T),$$

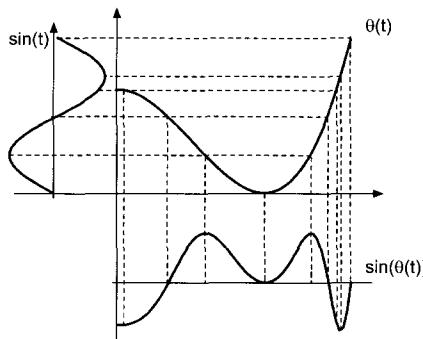


Figure 11.1 Time warping a sinewave by means of an arbitrary map $\theta(t)$.

where $\text{rem}(x, T)$ denotes the remainder of the integer division of x by T , have the same net effect on a T -periodic signal. More generally, we can time warp an arbitrary, aperiodic signal $s(t)$ via an arbitrary map, obtaining a signal

$$s_{tw}(t) = s(\theta(t))$$

whose waveshape and envelope may be completely different from the starting signal. If the map is invertible, i.e., one-to-one, then

$$s_{tw}(\theta^{-1}(t)) = s(t).$$

That is, at time $\tau = \theta^{-1}(t)$ the time warped signal attains the same amplitude value as that attained by the starting signal at time t .

Time warping transformations are useful for musical applications, e.g., for morphing a sound into a new one in the time domain.

11.2.2 Frequency Warping

Frequency warping is the frequency domain counterpart of time warping. Given a signal whose discrete-time Fourier transform (DTFT) is $S(\omega)$, we form the signal $s_{fw}(t)$ whose DTFT is

$$S_{fw}(\omega) = S(\theta(\omega)).$$

That is, the frequency spectrum of the frequency warped signal agrees with that of the starting signal at frequencies that are displaced by the map $\theta(\omega)$. If the map is invertible, then

$$S_{fw}(\theta^{-1}(\omega)) = S(\omega).$$

The frequency warped signal is obtained by computing the inverse DTFT of the warped frequency spectrum. In order to obtain a real warped signal from a real signal, the warping map must have odd parity, i.e.,

$$\theta(-\omega) = -\theta(\omega).$$

In order to illustrate the features of frequency warping, consider a periodic signal $s(t)$ whose frequency spectrum peaks at integer multiples of the fundamental frequency ω_0 . The frequency spectrum of the warped signal will peak at frequencies

$$\hat{\omega}_k = \theta^{-1}(k\omega_0).$$

The situation is illustrated in Fig. 11.2, where the original harmonic frequencies are represented by dots along the ordinate axis. The warped frequencies are obtained by drawing horizontal lines from the original set of frequencies to the graph of $\theta(\omega)$ and by reading the corresponding values of the abscissa. As a result, harmonically related partials are mapped into non-harmonically related ones. Furthermore, if the frequency warping map is not monotonically increasing, one obtains effects analogous to the foldover of frequencies. This is similar to that which is obtained from a phase vocoder in which the frequency bands are scrambled in the synthesis of the signal. However, the resolution and flexibility of the frequency warping method are generally much higher than that of the scrambled phase vocoder.

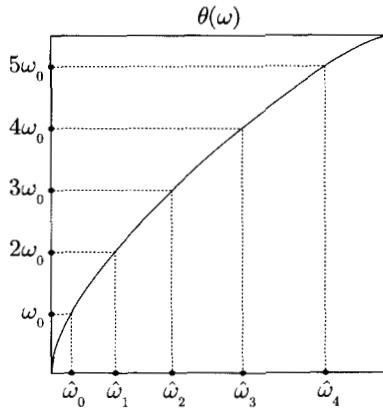


Figure 11.2 Frequency warping of a periodic signal: transformation of the harmonics into inharmonic partials.

Energy Preservation and Unitary Frequency Warping

By frequency warping a signal one dilates or shrinks portions of its frequency spectrum. As a result, the areas under the spectral characteristics are affected. Perceptually this results in an amplification of certain bands and an attenuation of other bands. This is depicted in Fig. 11.3 where the original narrow band spectrum of Fig. 11.3(a) is dilated obtaining the dotted curve shown in 11.3(b). In order to circumvent this problem, which causes an alteration of the relative energy levels of the spectrum, one should perform an equalization aimed at reducing the amplitude of dilated portions and increasing that of shrunk portions of the spectrum. Mathematically this is simply achieved, in the case where the warping map is increasing, by scaling the magnitude square of the DTFT of the warped signal by the derivative of the warping map. In fact, the energy in an arbitrary band $[\omega_0, \omega_1]$ is

$$E_{[\omega_0, \omega_1]} = \frac{1}{2\pi} \int_{\omega_0}^{\omega_1} |S(\omega)|^2 d\omega.$$

By the simple change of variable $\omega = \theta(\Omega)$ in the last integral we obtain

$$E_{[\omega_0, \omega_1]} = \frac{1}{2\pi} \int_{\Omega_0 = \theta^{-1}(\omega_0)}^{\Omega_1 = \theta^{-1}(\omega_1)} |S(\theta(\Omega))|^2 \frac{d\theta}{d\Omega} d\Omega = \frac{1}{2\pi} \int_{\Omega_0}^{\Omega_1} |\tilde{S}_{fw}(\Omega)|^2 d\Omega \quad (11.1)$$

where

$$\tilde{S}_{fw}(\omega) = \sqrt{\frac{d\theta}{d\omega}} S(\theta(\omega)) \quad (11.2)$$

is the DTFT of the scaled frequency warped signal. Equation (11.1) states the energy preservation property of the scaled warped signal in any band of the spectrum: the energy in any band $[\omega_0, \omega_1]$ of the original signal equals the energy of the warped signal in the warped band $[\theta^{-1}(\omega_0), \theta^{-1}(\omega_1)]$. Thus, the scaled frequency warping is a unitary operation on signals.

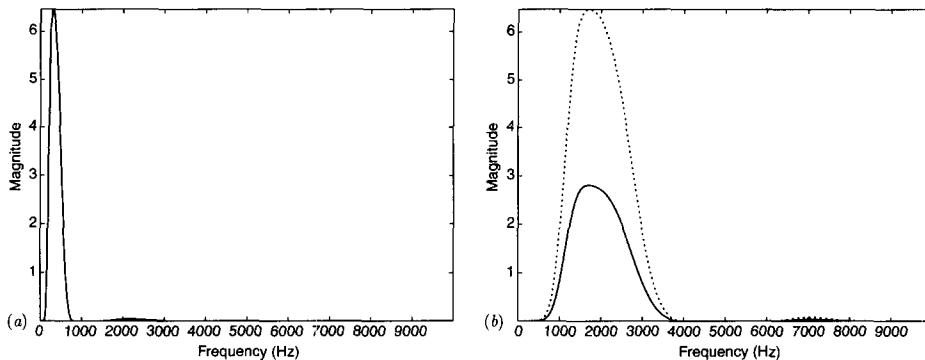


Figure 11.3 Frequency warping a narrow band signal: (a) original frequency spectrum and (b) frequency warped spectrum (dotted line) and scaled frequency warped spectrum (solid line).

11.2.3 Algorithms for Warping

In sections 11.2.1 and 11.2.2 we explored basic methods for time warping in the time domain and frequency warping in the frequency domain, respectively. However, one can derive time and frequency warping algorithms in crossed domains. It is easy to realize that time and frequency warping are dual operations. Once a time domain algorithm for frequency warping is determined, then a frequency domain algorithm for time warping will work the same way. This section contains an overview of techniques for computing frequency warping. The same techniques can be used for time warping in the dual domain. We start from the basic maps using the Fourier transform and end up with time-varying warping using allpass chains in dispersive delay lines.

Frequency Warping by Means of FFT

A simple way to implement the frequency warping operation on finite length discrete-time signals is via the FFT algorithm. Let

$$S\left(\frac{2\pi m}{N}\right) = \sum_{n=0}^{N-1} s(n)e^{-j\frac{2\pi n m}{N}}$$

denote the DFT of a length N signal $s(n)$. Consider a map $\theta(\omega)$ mapping the interval $[-\pi, \pi]$ onto itself and extend $\theta(\omega)$ outside this interval by letting

$$\theta(\omega + 2k\pi) = \theta(\omega) + 2k\pi, \quad k \text{ integer}$$

The last requirement is necessary in order to guarantee that the warped discrete time signal has 2π -periodic Fourier transform

$$S_{fw}(\omega + 2k\pi) = S(\theta(\omega + 2k\pi)) = S(\theta(\omega) + 2k\pi) = S(\theta(\omega)) = S_{fw}(\omega),$$

i.e., $S_{fw}(\omega)$ is the Fourier transform of a discrete-time signal. In order to obtain the frequency warped signal we would need to compute $S(\theta(\frac{2\pi m}{N}))$ and then perform the inverse Fourier transform. However, from the DFT we only know $S(\omega)$ at integer multiples of $\frac{2\pi}{N}$. The map $\theta(\omega)$ is arbitrary and $\theta(\frac{2\pi m}{N})$ is not necessarily a multiple of $\frac{2\pi}{N}$. However, we may approximate $\theta(\frac{2\pi m}{N})$ with the nearest integer multiple of $\frac{2\pi}{N}$, i.e., we can define the quantized map

$$\theta_q\left(\frac{2\pi m}{N}\right) = \frac{2\pi}{N} \text{ round} \left[\theta\left(\frac{2\pi m}{N}\right) \frac{N}{2\pi} \right].$$

The values $S(\theta_q(\frac{2\pi m}{N}))$ are known from the DFT of the signal and we can compute the approximated frequency warped signal by means of the inverse DFT:

$$s_{fw}(n) \approx \frac{1}{N} \sum_{m=0}^{N-1} S\left(\theta_q\left(\frac{2\pi m}{N}\right)\right) e^{j\frac{2\pi n m}{N}}.$$



Figure 11.4 Frequency warping by means of FFT: schematic diagram.

The diagram of the frequency warping algorithm via FFT is shown in Fig. 11.4. If the warping map is an increasing function, one can introduce the equalization factor as in (11.2) simply by multiplying $S(\theta_q(\frac{2\pi m}{N}))$ by the factor $\sqrt{\frac{d\theta}{d\omega}} \Big|_{\omega=\frac{2\pi m}{N}}$ before processing with the IFFT block. The FFT algorithm for warping is rather efficient, with a complexity proportional to $N \log N$. However, it has some drawbacks. The quantization of the map introduces distortion in the desired frequency spectrum, given by repetitions of the same value in phase and magnitude at near frequencies. These sum almost coherently and are perceived as beating components that have a slow amplitude decay. In frequency warping signals one must pay attention to the fact that the warped version of a finite length signal is not necessarily finite length. In the FFT-warping algorithm, components that should lie outside the analysis interval are folded back into this causing some echo artifacts. Furthermore, even if the original warping map is one-to-one, the quantized map is not and the warping effect cannot be undone without losses. The influence of the artifacts introduced by the FFT-warping algorithms may be reduced by zero-padding the original signal in order to obtain a larger value of N and, at the same time, a smaller quantization step for θ , at the expense of an increased computational cost.

Dispersive Delay Lines

In order to derive alternate algorithms for frequency warping [Bro65, OJ72], consider the DTFT (11.2) of the scaled frequency warped version of a causal signal $s(n)$:

$$\tilde{S}_{fw}(\omega) = \sqrt{\frac{d\theta}{d\omega}} S(\theta(\omega)) = \sqrt{\frac{d\theta}{d\omega}} \sum_{n=0}^{\infty} s(n) e^{-jn\theta(\omega)}. \quad (11.3)$$

The last formula is obtained by considering the DTFT of the signal $s(n)$, replacing ω with $\theta(\omega)$ and multiplying by $\sqrt{\frac{d\theta}{d\omega}}$. The warped signal $\tilde{s}_{fw}(k)$ is obtained from the inverse DTFT of $\tilde{S}_{fw}(\omega)$:

$$\tilde{s}_{fw}(k) = \text{IDTFT} \left[\tilde{S}_{fw}(\omega) \right] (k) = \sum_{n=0}^{\infty} s(n) \text{IDTFT} \left[\sqrt{\frac{d\theta}{d\omega}} e^{-jn\theta(\omega)} \right] (k). \quad (11.4)$$

Defining the sequences $\lambda_n(k)$ as follows,

$$\lambda_n(k) = \text{IDTFT} \left[\sqrt{\frac{d\theta}{d\omega}} e^{-jn\theta(\omega)} \right] (k) = \frac{1}{2\pi} \int_{-\pi}^{+\pi} \sqrt{\frac{d\theta}{d\omega}} e^{j[k\omega - n\theta(\omega)]} d\omega \quad (11.5)$$

we can put (11.4) in the form

$$\tilde{s}_{fw}(k) = \sum_{n=0}^{\infty} s(n) \lambda_n(k). \quad (11.6)$$

If we find a way of generating the sequences $\lambda_n(k)$, then we have a new algorithm for frequency warping, which consists of multiplying these sequences by the signal samples and adding the result. From (11.5) we have an easy way for accomplishing this since

$$\Lambda_n(\omega) = \text{DTFT}[\lambda_n](\omega) = \Lambda_{n-1}(\omega) e^{-j\theta(\omega)}, \quad (11.7)$$

with

$$\Lambda_0(\omega) = \sqrt{\frac{d\theta}{d\omega}}.$$

Notice that the term $e^{-j\theta(\omega)}$ has magnitude 1 and corresponds to an allpass filter. The sequence $\lambda_0(k)$ may be generated as the impulse response of the filter $\sqrt{\frac{d\theta}{d\omega}}$. The sequence $\lambda_n(k)$ is obtained by filtering $\lambda_{n-1}(k)$ through the allpass filter $e^{-j\theta(\omega)}$. This can be realized in the structure of Fig. 11.5 for computing the sequences $\lambda_n(k)$ as the impulse responses of a chain of filters. In order to perform warping it suffices to multiply each of the outputs by the corresponding signal sample and sum these terms together. The structure is essentially a delay line in which the elementary

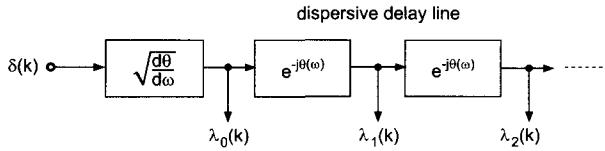


Figure 11.5 Dispersive delay line for generating the sequences $\lambda_n(k)$.

delays are replaced by allpass filters. Each of these filters introduces a frequency dependent group delay

$$\tau_G(\omega) = \frac{d\theta}{d\omega}.$$

The result is reminiscent of propagation of light in dispersive media where speed depends on frequency. For this reason this structure is called a dispersive delay line. What happens if we input a generic signal $y(k)$ to the dispersive delay line? The outputs $\hat{y}_n(k)$ are computed as the convolution of the input signal by the sequences $\lambda_n(k)$:

$$y(k) * \lambda_n(k) = \sum_r y(k-r)\lambda_n(r).$$

As a special case, for $k = 0$ and choosing as input the signal $s(k) = y(-k)$, which is the time-reversed version of $y(k)$, we obtain

$$\hat{y}_n(0) = \sum_r s(r)\lambda_n(r).$$

The last equation should be compared with (11.6) to notice that the summation is now over the argument of $\lambda_n(r)$. However, we can define the transposed sequences

$$\lambda_r^T(n) \equiv \lambda_n(r),$$

and write

$$\hat{y}_n(0) = \sum_r s(r)\lambda_r^T(n). \quad (11.8)$$

From (11.5) we have

$$\lambda_r^T(n) = \text{IDTFT} \left[\sqrt{\frac{d\theta}{d\omega}} e^{-jn\theta(\omega)} \right] (r) = \frac{1}{2\pi} \int_{-\pi}^{+\pi} \sqrt{\frac{d\theta}{d\omega}} e^{j[r\omega - n\theta(\omega)]} d\omega. \quad (11.9)$$

Suppose that the map $\theta(\omega)$ has odd parity, is increasing and maps π into π . Then we can perform in (11.9) the same change of variable $\Omega = \theta(\omega)$ as in (11.1) to obtain

$$\lambda_r^T(n) = \frac{1}{2\pi} \int_{-\pi}^{+\pi} \sqrt{\frac{d\theta^{-1}}{d\omega}} e^{j[n\omega - r\theta^{-1}(\omega)]} d\omega.$$

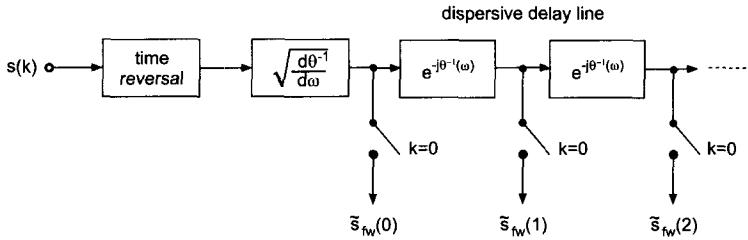


Figure 11.6 Computational structure for frequency warping.

As a result,

$$\Lambda_r^T(\omega) = \sqrt{\frac{d\theta^{-1}}{d\omega}} e^{jr\theta^{-1}(\omega)},$$

hence the transposed sequences $\lambda_r^T(n)$ have the same form as the sequences $\lambda_r(n)$ except that they are based on the inverse map $\theta^{-1}(\omega)$. Consequently, (11.8) is a formula for unwarping the signal. Furthermore, by exchanging the roles of $\theta(\omega)$ and $\theta^{-1}(\omega)$, (11.8) is also a valid algorithm for warping. The corresponding structure is shown in Fig. 11.6. The input signal is time reversed, then fed to the $\sqrt{\frac{d\theta^{-1}}{d\omega}}$ filter and to the dispersive delay line. The output of each filter is collected at time $k = 0$ by means of switches closing at that instant to form the scaled frequency warped sequence $\tilde{s}_{fw}(n)$. The structures in Figures 11.5 and 11.6 still present some computational problems. In general, the transfer functions involved are not rational. Furthermore, an infinite number of filters is needed for computing the transform. One can show that the only one-to-one map implementable by a rational transfer function is given by the phase of the first-order allpass filter

$$A(z) = \frac{z^{-1} - b}{1 - bz^{-1}}, \quad (11.10)$$

where $-1 < b < 1$. By varying the real parameter b in the allowed range, one obtains the family of Laguerre curves shown in Fig. 11.7. The curves with a negative value of the parameter are the inverses of those with a positive value, i.e., the inverse mapping $\theta^{-1}(\omega)$ corresponds to a sign reversal of the parameter. One can show that for causal signals the derivative $\sqrt{\frac{d\theta^{-1}}{d\omega}}$ can be replaced by the filter

$$\Lambda_0^T(z) = \frac{\sqrt{1 - b^2}}{1 + bz^{-1}}.$$

The structure of Fig. 11.6 includes a time reversal block and switches closing at time zero. It is clear that for a finite-length N signal one can equivalently form the signal $s(N - n)$ and close the switches at time $k = N$. Furthermore, by inspection of the structure, the required number M of allpass filters is approximately given by N times the maximum group delay, i.e.,

$$M \approx N \frac{1 + |b|}{1 - |b|}.$$

A larger number of sections would contribute little or nothing to the output signal.

The main advantage of the time-domain algorithm for warping is that the family of warping curves is smooth and does not introduce artifacts as opposed to the FFT-based algorithm illustrated in the above. Furthermore, the effect can be undone and structures for unwarping signals are obtained by the identical structure for warping provided that we reverse the sign of the parameter. In fact, the frequency warping algorithm corresponds to the computation of an expansion over an orthogonal basis giving rise to the Laguerre transform. Next we provide a simple MATLAB function implementing the structure of Fig. 11.6. The following M-file 11.1 gives a simple implementation of the Laguerre transform.

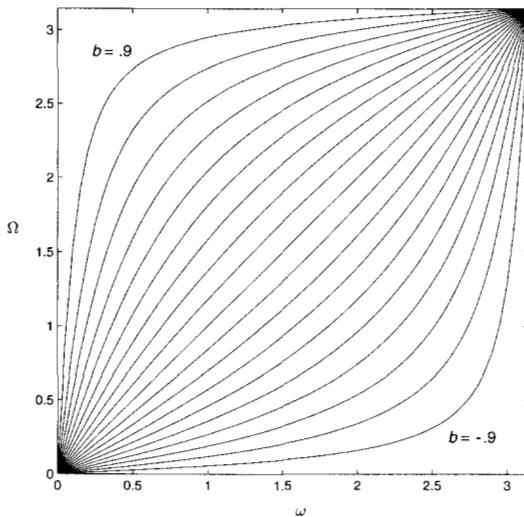


Figure 11.7 The family of Laguerre warping maps.

M-file 11.1 (lagt.m)

```
function y=lagt(x,b,M)
% computes M terms of the Laguerre transform y of the input x
% with Laguerre parameter b
N=length(x);
x=x(N:-1:1);      % time reverse input
% filter by normalizing filter lambda_0
yy=filter(sqrt(1-b^2),[1,b],x);
y(1)=yy(N);        % retain the last sample only
for k=2:M
% filter the previous output by allpass
yy=filter([b,1],[1,b],yy);
y(k)=yy(N);        % retain the last sample only
end
```

11.2.4 Short-time Warping and Real-time Implementation

The frequency warping algorithm based on the Laguerre transform illustrated in section 11.2.3 is not ideally suited to real-time implementation. Besides the computational cost, which is of the order of N^2 , each output sample depends on every input sample. Another drawback is that on a long signal the frequency-dependent delays cumulate to introduce large delay differences between high and low frequencies. As a result, the time organization of the input signal is destroyed by frequency warping. This can also be seen from the computational structure in Fig. 11.6, where subsignals pertaining to different frequency regions of the spectrum travel with different speeds along the dispersive delay line. At sampling time some of these signals have reached the end of the line, whilst other are left behind. For example, consider the Laguerre transform of a signal $s(n)$ windowed by a length N window $h(n)$ shifted on the interval $rM, \dots, rM + N - 1$. According to (11.6) we obtain

$$\tilde{s}_{fw}^{(r)}(k) = \sum_{n=rM}^{rM+N-1} h(n - rM) s(n) \lambda_n(k) = \sum_{n=0}^{N-1} x^{(r)}(n) \lambda_{n+rM}(k). \quad (11.11)$$

where

$$x^{(r)}(n) = h(n) s(n + rM).$$

The DTFT of (11.11) yields

$$\tilde{S}_{fw}^{(r)}(\omega) = e^{-jrM\theta(\omega)} \sum_{n=0}^{N-1} x^{(r)}(n) \Lambda_n(\omega) = e^{-jrM\theta(\omega)} \Lambda_0(\omega) X^{(r)}(\theta(\omega)). \quad (11.12)$$

From this we can see that the spectral contribution of the signal supported on $rM, \dots, rM + N - 1$ is delayed, in the warped signal, by the term $e^{-jrM\theta(\omega)}$, which introduces a largely dispersed group delay $M\tau_G(\omega)$. Approximations of the warping algorithm are possible in which windowing is applied in order to compute a short-time Laguerre transform (STLT) and, at the same time, large frequency-dependent delay terms are replaced by constant delays. In order to derive the STLT algorithm, consider a window $w(n)$ satisfying the perfect overlap-add condition

$$\sum_{r=-\infty}^{+\infty} w(n - rL) = 1, \quad (11.13)$$

where $L \leq N$ is an integer. This condition says that the superposition of shifted windows adds up to one. If $\tilde{s}_{fw}(n)$ denotes the Laguerre transform (11.6) of the signal $s(n)$, then we have identically:

$$\tilde{s}_{fw}(k) = \sum_{r=-\infty}^{+\infty} w(k - rL) \tilde{s}_{fw}^{(r)}(k) = \sum_{r=-\infty}^{+\infty} \sum_{n=0}^{+\infty} s(n) w(k - rL) \lambda_n(k). \quad (11.14)$$

By taking the DTFT of both sides of (11.14) one can show that

$$\tilde{S}_{fw}(\omega) = \sum_{r=-\infty}^{+\infty} e^{-jrL\omega} \frac{1}{2\pi} \int_{-\pi}^{+\pi} \Lambda_0(\Omega) S(\theta(\Omega)) W(\omega - \Omega) e^{jrL\Omega} d\Omega. \quad (11.15)$$

On the other hand, from (11.12) a delay compensated version of $\tilde{S}_{fw}^{(r)}(\omega)$ is

$$\tilde{S}_{fw}^{(r)}(\omega) = e^{-jr(L\omega - M\theta(\omega))} \tilde{S}_{fw}^{(r)}(\omega) = e^{-jrL\omega} \Lambda_0(\omega) X^{(r)}(\theta(\omega)) \quad (11.16)$$

which is the DTFT of the sequence

$$\hat{s}_{fw}^{(r)}(k) = \sum_{n=0}^{N-1} h(n)s(n + rM)\lambda_n(k - rL). \quad (11.17)$$

This equation defines the short-time Laguerre transform (STLT) of the signal $s(n)$. In order to select the proper integer M we need to study the term $X^{(r)}(\theta(\omega))$. One can show that

$$X^{(r)}(\theta(\omega)) = \frac{1}{2\pi} \int_{-\pi}^{+\pi} S(\Omega) H(\theta(\omega) - \Omega) e^{jrM\Omega} d\Omega. \quad (11.18)$$

We would like to approximate the integral in (11.15) by $\Lambda_0(\omega)X^{(r)}(\theta(\omega))$. Suppose that $H(\omega)$ is an unwarped version of $W(\omega)$, i.e., that

$$H(\omega) = \frac{d\theta^{-1}(\omega)}{d\omega} W(\theta^{-1}(\omega)) = |\Lambda_0^T(\omega)|^2 W(\theta^{-1}(\omega)). \quad (11.19)$$

By performing in (11.18) the change of variable $\Omega = \theta(\omega) + \theta(\alpha - \omega)$ we obtain

$$X^{(r)}(\theta(\omega)) = \frac{1}{2\pi} \int_{-\pi}^{+\pi} S(\theta(\omega) - \theta(\omega - \alpha)) W(\omega - \alpha) e^{jrM(\theta(\omega) - \theta(\omega - \alpha))} d\alpha. \quad (11.20)$$

Since $W(\omega)$ is a lowpass function, only the terms for $\alpha \approx \omega$ contribute to the last integral. Therefore, from (11.16) and (11.20) we conclude that the superposition of delay compensated versions of $\tilde{S}_{fw}^{(r)}(\omega)$ can be approximated as follows:

$$\hat{S}_{fw}(\omega) = \sum_{r=-\infty}^{+\infty} \tilde{S}_{fw}^{(r)}(\omega) \approx \sum_{r=-\infty}^{+\infty} e^{-jrL\omega} \frac{1}{2\pi} \int_{-\pi}^{+\pi} \Lambda_0(\alpha) S(\theta(\alpha)) W(\omega - \alpha) e^{jrM\theta(\alpha)} d\alpha. \quad (11.21)$$

Equation (11.21) should be compared with (11.15). A linear approximation of $\theta(\alpha)$ is

$$\theta(\alpha) = \theta'(0)\alpha + O(\alpha^3) = \frac{1+b}{1-b}\alpha + O(\alpha^3). \quad (11.22)$$

One can show that this is a fairly good approximation for $|\alpha| < \frac{1-b}{2}\pi$. In this

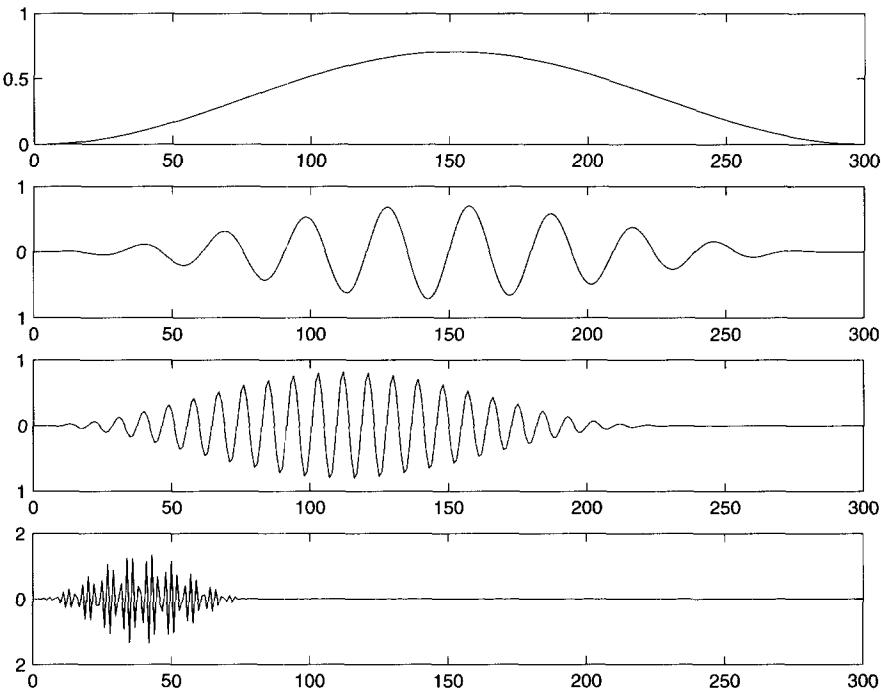


Figure 11.8 Short-time warping: different length of warped signals from low to high frequencies (top to bottom).

frequency range, if we select

$$M \approx \frac{1-b}{1+b}L \quad (11.23)$$

then

$$\hat{S}_{fw}(\omega) \approx \tilde{S}_{fw}(\omega),$$

i.e., the overlap-add of STLT components well approximates the Laguerre transform. In other words, an approximate scheme for computing the Laguerre transform consists of taking the Laguerre transform of overlapping signal frames windowed by the unwarped window $h(n)$ and overlap-adding the result, as shown in Fig. 11.9. This method allows for a real-time implementation of frequency warping via the Laguerre transform. It relies on the linear approximation (11.22) of the phase of the allpass, valid for the low-frequency range. An important issue is the choice of the window $w(n)$. Many classical windows, e.g., rectangular, triangular, etc., satisfy condition (11.13). However, (11.21) is a close approximation of the Laguerre transform only if the window sidelobes are sufficiently attenuated. Furthermore, the unwarped version (11.19) of the window can be computed via a Laguerre transform with the

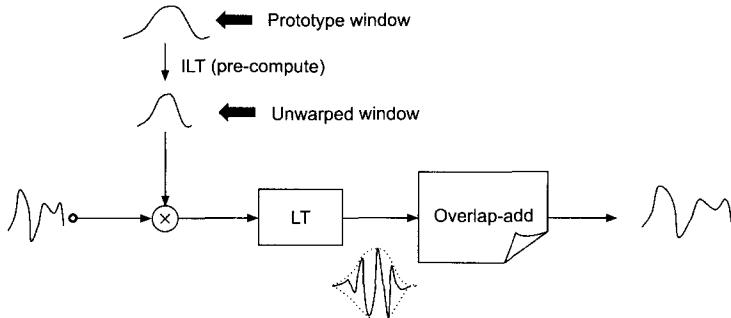


Figure 11.9 Block diagram of the approximate algorithm for frequency warping via overlap-add of the STLT components. The block LT denotes the Laguerre transform and ILT its inverse.

normalizing filter $\Lambda_0^T(\omega)$ removed. In principle $h(n)$ has infinite length. However, the inverse Laguerre transform of a lowpass window $w(n)$ has essential length

$$N \approx \left. \frac{d\theta^{-1}}{d\omega} \right|_{\omega=0} N_w = \frac{1-b}{1+b} N_w.$$

In order to avoid artifacts in the multiplication of the signal by the window we are interested in windows whose Laguerre transform essentially is a dilated or stretched version of the window itself. This property turns out to be approximately well satisfied by the Hanning window

$$w(n) = \begin{cases} \frac{1}{2} \left(1 - \cos \frac{2\pi n}{N_w} \right) & n = 0, 1, \dots, N_w - 1 \\ 0 & \text{otherwise} \end{cases}.$$

The choice of the length N_w is arbitrary. Furthermore, the Hanning window satisfies (11.13) for any L integer submultiple of N_w . Long windows tend to better approximate pure frequency warping. However, both response time and computational complexity increase with the length of the window. Moreover, the time-organization destruction effect is more audible using extremely long windows. The integer L controls the overlap $N_w - L$ of the output warped frames. When warping with a positive value of the parameter b one should select a considerable overlap, e.g., $N_w = 5L$, in order to avoid amplitude distortion of the high-frequency components, which, in this case, are more concentrated in the Laguerre domain, as shown in Fig. 11.8. Finally, the integer M fixing the input frames overlap is obtained by rounding the right-hand side of (11.23). Next we provide a simple M-file 11.2 implementing frequency warping by means of STLT overlap-add. The function gives a simple implementation of frequency warping via short-time Laguerre transform.¹

¹The function lugtun is the same as lagt reported in section 11.2.3, except that the line `yy=filter(sqrt(1-b^2),[1,b],x);` is replaced by the line `yy=x;` in order to compute the non-normalized Laguerre transform.

```
M-file 11.2 (winlagt.m)
function sfw=winlagt(s,b,Nw,L)
% Frequency warping via STLT of the signal s with parameter b,
% output window length Nw and time-shift L
w=L*(1-cos(2*pi*(0:Nw-1)/Nw))/Nw;           % normalized Hanning window
N=ceil(Nw*(1-b)/(1+b));                      % length of unwarped window h
M=round(L*(1-b)/(1+b));                      % time-domain window shift
h=lagtun(w,-b,N); h=h(:)                      % unwarped window
Ls=length(s);                                  % pad signal with zeros
K=ceil((Ls-N)/M);                            % to fit an entire number
s=s(:); s=[s ; zeros(N+K\ast M-Ls,1)];        % of windows
Ti=1; To=1;                                    % initialize I/O pointers
Q=ceil(N*(1+abs(b))/(1-abs(b)));            % length of Laguerre transform
sfw=zeros(Q,1);                                % initialize output signal
for k=1:K
    yy=lagt(s(Ti:Ti+N-1).*h,b,Q);          % Short-time Laguerre transf.
    sfw(To:end)=sfw(To:end)+yy;              % overlap-add STLT
    Ti=Ti+M; To=To+L;                      % advance I/O signal pointers
    sfw=[sfw; zeros(L,1)];                  % zero pad for overlap-add
end
```

11.2.5 Time-varying Frequency Warping

Suppose that each frequency-dependent delay element in the structure of Fig. 11.5 has its own phase characteristics $\theta_k(\omega)$ and suppose that we remove the scaling filter. Accordingly, the outputs of the structure are the sequences

$$\psi_n(k) = a_1(k) * a_2(k) * \dots * a_n(k)$$

with $\psi_0(k) = \delta(k)$, obtained by convolving the impulse responses $a_m(k)$ of the allpass filters

$$A_m(z) = \frac{z^{-1} - b_m}{1 - b_m z^{-1}}.$$

Hence the z-transforms of the sequences $\psi_n(k)$ are

$$\Phi_n(z) = \prod_{k=1}^n \frac{z^{-1} - b_m}{1 - b_m z^{-1}}$$

and their DTFT is

$$\Phi_n(\omega) = \prod_{k=1}^n e^{-j\theta_k(\omega)} = e^{-j\Theta_n(\omega)},$$

where

$$\Theta_n(\omega) = \sum_{k=1}^n \theta_k(\omega)$$

is the sign-reversed cumulative phase of the first n delay elements. By multiplying each signal sample $s(n)$ by the corresponding sequence $\varphi_n(k)$ we obtain the signal

$$s_{tvfw}(k) = \sum_{n=0}^{+\infty} s(n)\varphi_n(k),$$

whose DTFT is

$$S_{tvfw}(\omega) = \sum_{n=0}^{+\infty} s(n)e^{-j\Theta_n(\omega)}.$$

Note that this is an important generalization of (11.3) in which the phase terms are not integer multiples of each other. In the special case where all the delays are equal we have $\theta_k(\omega) = \theta(\omega)$ and $\Theta_n(\omega) = n\theta(\omega)$. If we suppose that the delays are equal in runs of N , then signals of finite length N , supported on the intervals $(r-1)N, \dots, rN-1$ are frequency warped according to distinct characteristics. For the same reason, signal samples falling in these intervals are differently warped. Portions of the signal falling in two adjacent intervals are warped in a mixed way. More generally, one can have a different delay for each signal sample. This results in a time-varying frequency warping [EC99, EC00]. From a musical point of view one is often interested in slow and oscillatory variations of the Laguerre parameter, as we will discuss in section 11.3. It is possible to derive a computational structure for time-varying warping analogous to that reported in Fig. 11.6. This is obtained by considering the sequences $\psi_n(k)$ whose z-transforms satisfy the following recurrence:

$$\begin{aligned}\Psi_0(z) &= \frac{1}{1 - b_1 z^{-1}} \\ \Psi_n(z) &= H_n(z)\Psi_{n-1}(z)\end{aligned}$$

where

$$H_n(z) = \frac{1 - b_n b_{n+1}}{1 - b_{n-1} b_n} \frac{z^{-1} - b_{n-1}}{1 - b_{n+1} z^{-1}}$$

and $b_0 = 0$. This set of sequences plays the same role as the transposed sequences (11.9) in the Laguerre expansion. However, the sequences $\varphi_n(k)$ and $\psi_n(k)$ are not orthogonal, rather, they are biorthogonal, i.e.,

$$\sum_{k=0}^{+\infty} \varphi_n(k)\psi_m(k) = \delta(n-m).$$

Consequently, our time-varying frequency warping scheme is not a unitary transform of the signal, hence it does not verify the energy preservation property (11.1). However, one can show that this is a complete representation of signals. Hence the time-varying frequency warping is an effect that can be undone without storing the original signal. The modified structure for computing time-varying frequency warping is reported in Fig. 11.10. In order to preserve the same direction of warping as

in the fixed parameter Laguerre transform, the sign of the parameter sequence must be reversed, which is equivalent to exchanging the roles of $\theta_n(\omega)$ and $\theta_n^{-1}(\omega)$. The inverse structure can be derived in terms of a tapped dispersive delay line based on $\Phi_n(\omega)$ with the warped signal samples used as tap weights. Next we provide a simple M-file 11.3 implementing the structure of Fig. 11.10. The function gives a simple implementation of the variable parameter generalized Laguerre transform.

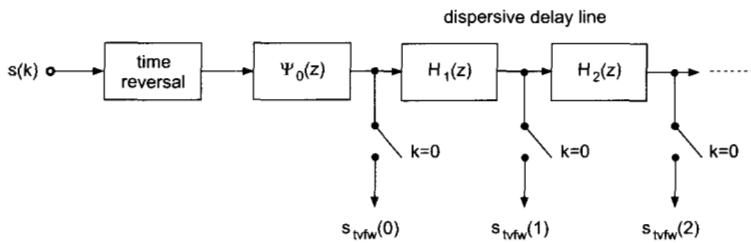


Figure 11.10 Structure for computing time-varying frequency warping via generalized Laguerre transform with variable parameter.

M-file 11.3 (lagtbvar.m)

```

function y=lagtbvar(x,b,M)
% computes coefficients y of biorthogonal Laguerre expansion of x
% using variable parameters b(k) where b is a length M array
N=length(x);
yy=x(N:-1:1); % time reverse input
y=zeros(1,M);
yy=filter(1,[1, b(1)],yy); % filter by psi_0(z)
y(1)=yy(N); % retain the last sample only
% filter by H_1(z)(unscaled, b to -b)
yy=filter([0,1],[1, b(2)],yy);
y(2)=yy(N)*(1-b(1)*b(2)); % retain the last sample only and scale
for k=3:M
    % filter by H_(k-1)(z)(unscaled, b to -b)
    yy=filter([b(k-2),1],[1, b(k)],yy);
    y(k)=yy(N)*(1-b(k-1)*b(k)); % retain the last sample only and scale
end

```

Time-varying frequency warping has a fast approximate algorithm whose block diagram is reported in Fig. 11.11. The scheme is similar to the overlap-add method derived for the Laguerre transform and is shown in Fig. 11.9. However, due to the time-varying aspect, the inverse time-varying warping of the prototype window must be computed for each input frame.

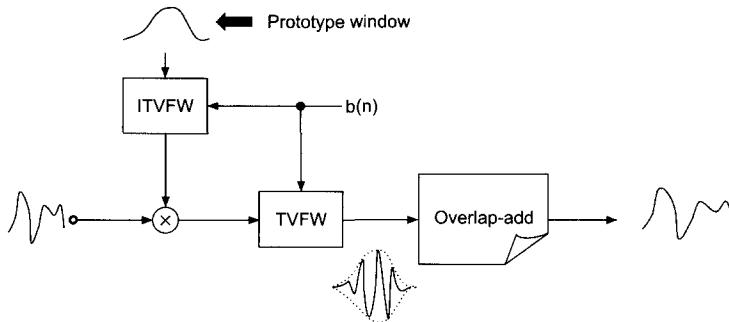


Figure 11.11 Block diagram of the approximate algorithm for time-varying frequency warping. The blocks TVFW and ITVFW respectively denote time-varying frequency warping and its inverse.

11.3 Musical Uses of Warping

In this section we describe a few applications of warping in music. As already pointed out, many aspects and properties of warping musical signals are still to be explored and many results of this section may be deemed as experimental. Applications that will be discussed range from accurate pitch-shifting of inharmonic sources and inharmonization of harmonic sources, to feature and transient extraction, vibrato editing and morphing.

11.3.1 Pitch Shifting Inharmonic Sounds

The sounds from a large class of instruments are inherently inharmonic. The spacing of the frequencies of the partials is not uniform. In piano sounds, in the low register, the displacement of the partials from the harmonics becomes more and more apparent as we move towards the lower end of the keyboard. In Fig. 11.12 we report data (\times marks) extracted from a low-pitch piano tone (≈ 27 Hz). These represent the differences between the frequency of a partial and that of the next one. If the sound were harmonic, one should observe a flat distribution of points aligned on the pitch frequency. On the contrary, one observes that the spacing between the partials increases with the order of the overtones. The distribution of the partials can be closely matched to the derivative of a Laguerre curve. This can be obtained by means of an optimization of the parameter b in (11.10). It turns out that the absolute value of the optimum Laguerre parameter decreases as we move from lower to higher tones. This means that the warping curve becomes more and more linear, as can be seen from Fig. 11.7. By frequency warping the original piano tone with the inverse of the fitted Laguerre map one transforms the originally inharmonic partials into a set of harmonic partials. As a result of warping the fundamental frequency, the pitch of the resulting tone will be higher. Vice versa, by warping by a Laguerre map with a small positive value of the parameter one decreases pitch and increases the degree of inharmonicity. This gives us a method for pitch shifting piano tones

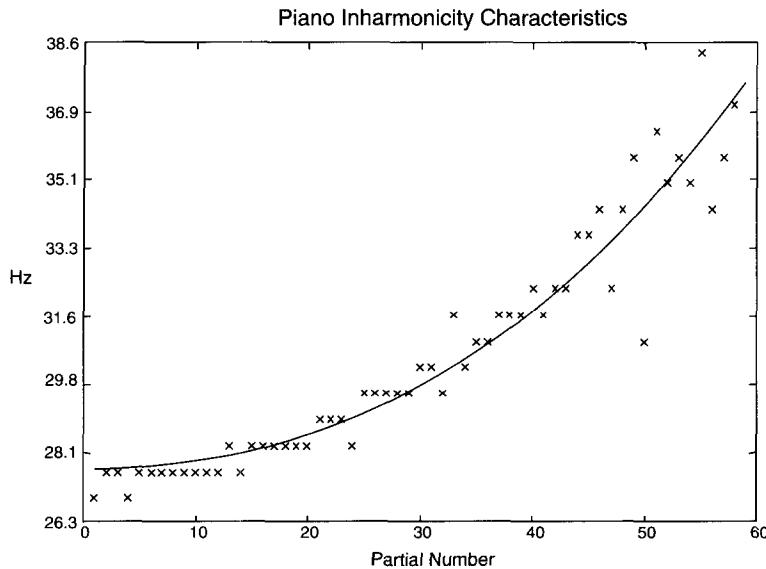


Figure 11.12 Inharmonicity characteristics of a 27 Hz piano tone: data are marked by x and the solid curve represents the optimum Laguerre difference curve fitting the data.

that is particularly accurate in matching the inharmonicity of lower tones. Given a piano tone one can determine the value and the sign of the warping parameter in order to transform it to a lower or higher tone. Specifically, suppose that the fundamental frequency is f_0 and that the desired frequency is \hat{f}_0 . In terms of the normalized frequency ω , with a sampling rate f_S , we have, respectively, $\omega_0 = \frac{2\pi f_0}{f_S}$ and $\hat{\omega}_0 = \frac{2\pi \hat{f}_0}{f_S}$. As remarked in section 11.2.2 the new normalized fundamental frequency after warping is $\hat{\omega}_0 = \theta^{-1}(\omega_0)$. One can show that

$$\theta^{-1}(\omega) = 2 \arctan \left(\frac{1-b}{1+b} \tan \frac{\omega}{2} \right),$$

hence we can determine the required value of b as follows:

$$b = \frac{\tan \frac{\pi f_0}{f_S} - \tan \frac{\pi \hat{f}_0}{f_S}}{\tan \frac{\pi f_0}{f_S} + \tan \frac{\pi \hat{f}_0}{f_S}}. \quad (11.24)$$

For inharmonic sounds, pitch shifting by frequency warping is more accurate than conventional algorithms based on proportional scaling of fundamental frequency and overtones. In fact, the warping characteristics can be ultimately justified by means of a physical model of stiff strings or membranes [VS94, TEC97]. It is quite striking that the Laguerre characteristics match those of piano tones for a large range. Therefore one obtains accurate pitch-shifting and the inharmonicity law by pure frequency warping. Otherwise one should resort to a combination of conventional

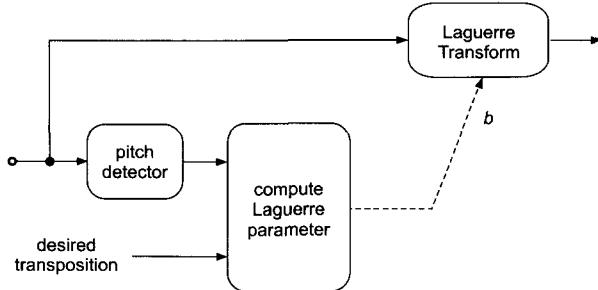


Figure 11.13 Block diagram of inharmonic sounds pitch shifter.

pitch-shifting and warping. The block diagram of a pitch shifter for inharmonic sounds based on the Laguerre transform is shown in Fig. 11.13. Frequency warping can also be used in conjunction with proportional pitch-shifting algorithms to pitch-shift harmonic signals. These techniques usually yield a rational alteration of the pitch and one needs to improve their resolution. Also, ratios other than simple ratios with small integers in both numerator and denominator are costly from a computational point of view. By frequency warping the proportionally pitch-shifted signal with a small absolute value of the warping parameter one can introduce a small incremental pitch-shifting operation, which, when added to the rational pitch-shifting operation, provides a pitch closer or equal to the desired pitch. At the same time, the inharmonicity introduced is not perceptually relevant due to the small value of the warping parameter.

11.3.2 Inharmonizer

Among the new effects introduced by frequency warping there is the inharmonizer. This effect is obtained by frequency warping an original harmonic sound with a large absolute value (≈ 0.5) of the parameter. The resulting sound is enriched by inharmonic partials, maps of the original harmonic partials, as discussed in section 11.2.2. Notice that both pitch and duration of the original sound are altered by warping. In fact, frequency warping stretches or shrinks the width of the peaks centered on the partial frequencies. As a result, the amplitude envelopes of the partials are altered. In the first approximation they are simply time-scaled. In order to restore the original pitch and duration one can resort to resampling techniques. At conventional sampling rates (20–44 kHz) the fundamental frequency of a large class of sounds from natural instruments falls into the low frequency portion of the axis. In that region the warping map is approximately linear with coefficients which are the derivative of the map in $\omega = 0$. This is also the amount by which the duration of the signal is scaled. This makes it possible to achieve pitch and duration rescaling by a single resampling operation. In many cases the inharmonizer effect introduces interesting detuning of the higher partials, transforming, for example, a trumpet sound into a bell-like sound or a guitar sound into a piano-like sound.

The inharmonizer can also be used in physical model synthesis, e.g., as a Karplus-Strong post-processing block or embedded in the delay line, in order to model inharmonicity due to dispersive propagation in stiff media.

11.3.3 Comb Filtering+Warping and Extraction of Excitation Signals in Inharmonic Sounds

As previously pointed out, by frequency warping the original piano tone with the inverse of the fitted Laguerre map, one transforms the originally inharmonic partials into a set of harmonic partials. This property can be exploited in order to extract the hammer noise from piano sounds. In fact, the audible effect of the hammer noise lies in areas of the frequency spectrum that are not masked by the partials, i.e., in between the partials. It is easy to build a comb filter based on the harmonics of the transformed piano sound. In fact, given a narrow-band lowpass filter with frequency response $H(\omega)$, the frequency response $H(\omega P)$, where P is the period of the signal expressed in number of samples is a comb filter adjusted to the harmonics. This filter is obtained by inserting $P - 1$ zeros in the filter coefficients. Likewise, if $G(\omega)$ is a high-pass filter, the filter $G(\omega P)$ will select all the frequency bands that lie in between the harmonics. In order to obtain the piano hammer noise it suffices to un warp the signal in order to regularize the partials into harmonics, determine the transformed pitch, filter with $G(\omega P)$ and apply frequency warping to re-obtain the inharmonic distribution. In the present case it is more convenient to prewarp the filters rather than the signals. However, in a more general setting where the inharmonic signal components are analyzed by means of pitch-synchronous wavelets [Eva93, Eva94], which include downsampling operations, it can be shown that it is more convenient to warp the signal [EC97, EC98a, EC98b]. The block diagram of a tuned warped comb filter is shown in Fig. 11.14.

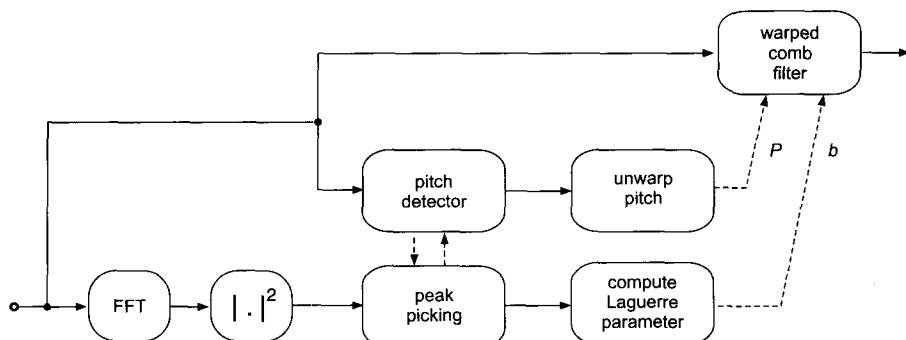


Figure 11.14 Block diagram of tuned warped comb structure for extracting partials or excitation noise from inharmonic sounds.

11.3.4 Vibrato, Glissando, Trill and Flatterzunge

Vibrato can be generated by means of time-varying frequency warping, by using an oscillating sequence of parameters b with low amplitude and frequency. For small values of the warping parameter, the warping curve only slightly deviates from the linear map and the harmonic structure of the signal is essentially preserved, while pitch-shifting is the only perceptually relevant effect. This is especially true when the parameter law is oscillatory so that the harmonics fluctuate around their original frequency. This allows us to introduce dynamic pitch fluctuations in natural or synthetic sounds, which can be directly controlled by the warping parameter sequence according to equation (11.24). In particular, one can use a sinusoidal LFO as a control parameter generator to insert very natural vibrato. Both the frequency and amplitude of the oscillator can be changed at will, i.e., to synchronize the effect to the amplitude envelope of the signal or to include random fluctuations. Trill and rapid fluctuations of the pitch can be obtained by means of a square wave LFO. By mixing pitch-modulated versions of the sound with the original signal one can obtain effects similar to phasing, flanging and chorusing. By frequency warping a flute sound using random noise or random amplitude square wave as parameter sequences one obtains interesting effects typical of Flatterzunge. As another example, glissando can be inserted by means of an increasing or decreasing sequence of parameters. A general structure based on mixed independent time-varying warping channels for computing the above effects is shown in Fig. 11.15. In much the same way, one can edit sounds containing vibrato or any pitch modulation in order to reduce or remove this effect. It suffices to extract the pitch fluctuation law from the sound by means of a pitch detection algorithm or by tracking the partials in the spectrogram of the sound. From this law one can obtain the law of variation of the parameter b and by applying the time-varying frequency warping algorithm with a reversed sign sequence of parameters, one can counteract the pitch modulation effect [EC00].

11.3.5 Morphing

Accurate spectral morphing requires arbitrary maps of the frequency axis in order to transform the partials of one sound into the partials of another sound. The FFT warping algorithm illustrated in section 11.2.3 can be employed with simplicity to perform this task. However, since invertibility is not an issue, versions of the Laguerre transform based on higher order allpass filters can be employed as well. In order to determine the suitable warping map one can use a peak-picking algorithm in the frequency domain to detect the partials of both the original and desired sound. Simple morphing examples can be computed using the structure shown in Fig. 11.16. A set of points on an initial-final frequency plane is determined, which can be interpolated to produce a smooth warping curve. As an example one can eliminate the even harmonics in a voiced sound by mapping these into odd harmonics. Realistic morphing also requires amplitude scaling of the partials. This corresponds to a simple filtering operation on the signal. Morphing can also be performed as a dynamic operation by means of time-varying frequency warping using a sequence of maps.

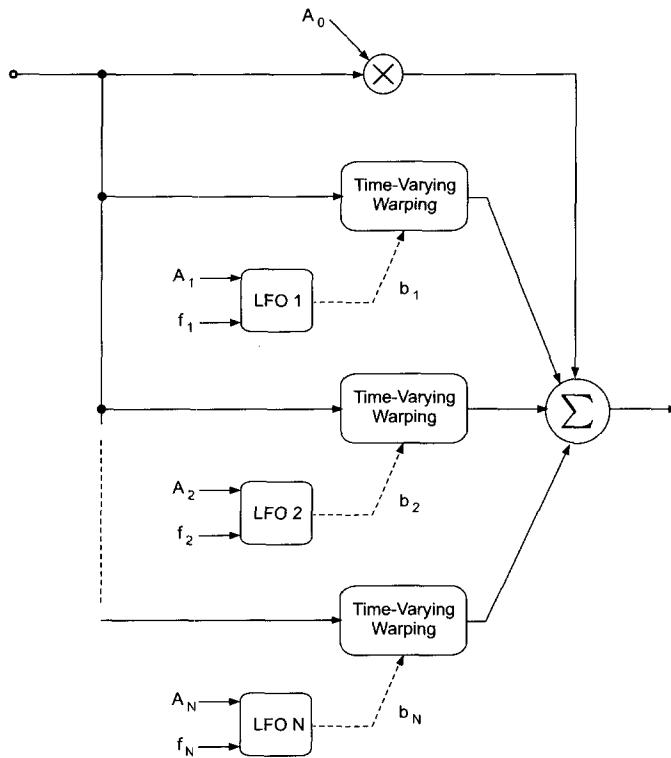


Figure 11.15 Block diagram for computing vibrato, trill, chorus-like, phasing-like or flange-like effects. For Flatterzunge we add random noise to the LFOs. For glissando the LFOs are replaced by envelope generators.

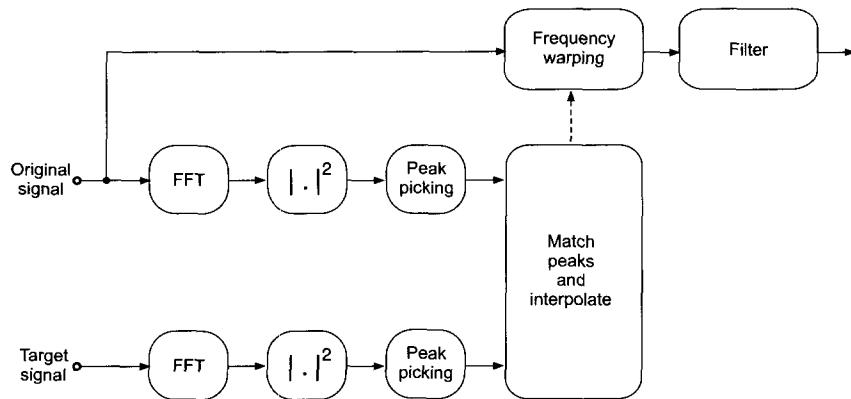


Figure 11.16 Simple diagram for computing morphing via frequency warping.

11.4 Conclusion

In this chapter we introduced a class of digital audio effects based on frequency warping techniques of recent interest in musical applications. The deformation of the frequency axis, whether static or dynamic, introduces a new point of view and new tools for processing sounds. This transformation allows us to insert or edit vibrato, trill, Flatterzunge and glissando, adding controlled expression to static sounds. Harmonic sounds can be mapped into inharmonic sounds, introducing fine partial detuning to color them. Frequency warping also provides a concerned or model-based method for pitch-shifting inherently inharmonic sounds such as piano and drums sounds. Mixing independent time-varying warping channels achieves interesting generalizations of flanging, chorusing and phasing effects. An efficient algorithm based on the short-time Laguerre transform makes frequency warping computable in real-time. Since frequency warping is at present fairly unexploited in musical contexts we encourage musicians and sound engineers to experiment with this appealing technique.

Bibliography

- [Bro65] P.W. Broome. Discrete orthonormal sequences. *J. Assoc. Comput. Machinery*, 12(2):151–168, 1965.
- [EC97] G. Evangelista and S. Cavaliere. Analysis and regularization of inharmonic sounds via pitch-synchronous frequency warped wavelets. In *Proc. International Computer Music Conference*, pp. 51–54, Thessaloniki, Greece, September 1997.
- [EC98a] G. Evangelista and S. Cavaliere. Discrete frequency warped wavelets: theory and applications. *IEEE Trans. on Signal Processing*, special issue on Theory and Applications of Filter Banks and Wavelets, 46(4):874–885, April 1998.
- [EC98b] G. Evangelista and S. Cavaliere. Frequency warped filter banks and wavelet transform: a discrete-time approach via Laguerre expansions. *IEEE Trans. on Signal Processing*, 46(10):2638–2650, October 1998.
- [EC99] G. Evangelista and S. Cavaliere. Time-varying frequency warping: results and experiments. In *Proc. DAFX-99 Digital Audio Effects Workshop*, pp. 13–16, Trondheim, 1999.
- [EC00] G. Evangelista and S. Cavaliere. Audio effects based on biorthogonal time-varying frequency warping. *EURASIP Journal on Applied Signal Processing*, 1(1):27–35, March 2001.
- [Eva93] G. Evangelista. Pitch synchronous wavelet representations of speech and music signals. *IEEE Trans. on Signal Processing*, special issue on Wavelets and Signal Processing, 41(12):3313–3330, December 1993.

- [Eva94] G. Evangelista. Comb and multiplexed wavelet transforms and their applications to signal processing. *IEEE Trans. on Signal Processing*, 42(2):292–303, February 1994.
- [OJ72] A.V. Oppenheim and D.H. Johnson. Discrete representation of signals. *Proc. IEEE*, 60:681–691, June 1972.
- [TEC97] I. Testa, G. Evangelista, and S. Cavalieri. A physical model of stiff strings. *Proc. of the Institute of Acoustics (Internat. Symp. on Music and Acoustics, ISMA '97)*, Vol. 19: Part 5 (1997) Book 1, pp. 219–224, Edinburgh, August 1997.
- [VS94] S.A. Van Duyne and J.O. Smith. A simplified approach to modeling dispersion caused by stiffness in strings and plates. In *Proc. International Computer Music Conference*, pp. 407–410, 1994.

Chapter 12

Control of Digital Audio Effects

T. Todoroff

12.1 Introduction

The control of parameters of sound processing algorithms is an important issue, which cannot be overlooked. However cleverly programmed, an algorithm in itself has rarely been a useful tool in the hands of a musician, a sound engineer or a composer ... unless he also happens to be a computer programmer and has the ability to design his own control strategies.

Control, in the broad meaning of the word, encompasses every possible method available to the user for accessing the various parameters of a digital audio effect. It embraces all traditional computer user interfaces, from command-line instructions typed on a computer keyboard to complex window-based GUIs (Graphical User Interface) controlled with the mouse. Control also includes specially designed musical interfaces, mainly MIDI (Musical Instrument Digital Instrument) triggering devices mimicking various aspects of traditional instruments and widely commercialized by the music industry: organ-like or piano-like keyboards, drum pads, string and wind instrument controllers, as well as common studio controllers like rotary potentiometers, faders and push-buttons. And if all this does not give enough possibilities, there is a whole range of more or less experimental devices that have been designed to fulfill more specific needs. They include the radio baton, electromagnetic and ultrasound localization systems, power gloves, complete virtual reality environments, video image analysis systems and just about every possible sensor.

Features extracted from a sound may also be used to control parameters. The two most commonly found feature extractors, envelope and pitch followers, were already widely used in older analogue equipment like modular synthesizers, vocoders or guitar controllers. But partial tracking, spectral envelope tracking, centroid tracking,

voiced/unvoiced and silence/sound detection can also prove very useful. Often the feature extractors are so deeply integrated into some audio effects that they cannot be separated. Consider some of the effects described in the previous chapters: non-linear dynamics processors, vocoders, pitch-synchronous processes, hybridization, etc.

Finally, one can design control algorithms whose only task is to send parameter values to the sound processing algorithm. Stochastic functions, cellular automata, genetic algorithms, physical models or any time-varying function might be used. In this case, the control algorithm in turn has to be controlled by the user. This opens up an endless list of potentially interesting combinations.

But, first, we will have a look at the general context which guides all control approaches and discuss the important mapping issues. They address the search for intuitive and effective ways to translate user intentions into parameter values needed for sound processing algorithms. Control is such a broad topic that we have to limit the scope of this chapter and try to give a brief overview. We will therefore put an emphasis on real-time systems and on gestural control.

12.2 General Control Issues

Control is not only about technology, there is a strong human side involved. Control of our actions in daily life relies on many concurrent processes where feedback loops play an important part. We use the many signals coming from the body to continuously adjust the way we control our muscles. One cannot speak properly without hearing one's voice, hence the great difficulties deaf people face when trying to learn how to speak. We use sensory information that gives us hints about position of jaw, tongue, lips, etc. But it is interesting to note that most training strategies developed for deaf people make use of visual feedback to compensate for the missing sense, showing the importance of feedback and demonstrating at the same time that one can learn to implement alternative feedback mechanisms. The learning process plays a leading role as we can gradually use our past experiences to predict the effects signals sent to the muscles will have. And as more basic feedback patterns become subconscious (we almost never think consciously about how we articulate when talking in our native language nor about how we are walking), we can devote more attention to controlling the subtle variations. We have to keep this in mind when designing control interfaces for Digital Audio Effects. Figure 12.1 shows a simplified view of the various feedback mechanisms applied to playing a computer instrument.

With such complex processes taking place, it is little wonder that we accept as normal the fact that traditional performers learn their instruments for many years before performing in front of an audience. On the other hand, strangely enough, we expect computer interfaces to give us access to the full expressive power of an instrument after only a few hours or days of training. Understanding this discrepancy, we should accept some trade-off and choose between implementing either an easy-to-play instrument or a more difficult one, where training will be rewarded by

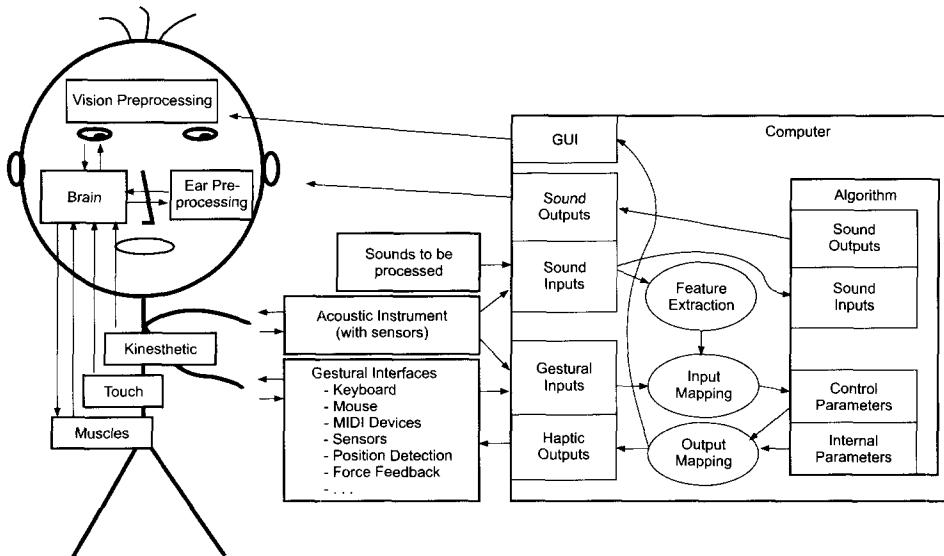


Figure 12.1 Schematic control feedback loop.

wider expressive capabilities. The best choice depends on the context. While it is normal for a performer to practise his instrument before a concert, one should not expect a visitor of an interactive sound installation to train for several days before being allowed in.

Therefore, the most important thing to keep in mind when designing a user interface for a specific audio effect is to think about the end user. It may sound trivial but, as it facilitates their only link with the underlying algorithm, the chosen interface should not only implement an effective way of controlling the sound processing, but should also focus on understanding and fulfilling the needs of the intended user. This means evaluating his acquired knowledge and practice and building on it. It is therefore easier to design an interface when emulating an acoustical instrument or an existing analog effect than when devising a totally new one. The large number of digital audio products sold in hardware boxes which feature almost identical control buttons as their analog ancestors illustrates this.

As a consequence, different control strategies should be applied to the same sound-processing algorithm [Tod95] depending on whether it will be used by a sound engineer for a post-production task, a musician during a live performance, a composer in his studio, a dancer on stage or an audience visiting a sound installation.

12.3 Mapping Issues

Mapping is a way to transform one representation into another. If we look again at Fig. 12.1, we see two categories of mapping:

- Input mapping translates user's actions into parameter values needed to drive the sound processing algorithms. This is the most commonly accepted meaning of the word "mapping".
- Output mapping does the reverse, representing the algorithms parameters in a way that makes sense to the user. This aspect of mapping is obviously most often related to visual feedback, but it could also be used to give a tactile or haptic feedback. For instance, the resistance of a gestural interface could increase proportionally to one parameter. This can be compared to what happens when a musician pulls the string of an instrument: the further he pulls it away from the body of the instrument, the higher the resistance to pulling it further increases.

Therefore, some forms of mapping are needed whatever type of control is used (GUI, gestural controllers, feature extraction or algorithmic control). The more it takes perceptive aspects into account, the better it will fulfill its role in making the control loops work intuitively and effectively.

In the case of traditional instruments, it seems that the mapping between gestural parameters and sound parameters is usually direct and simple: the position of the finger on the piano directly maps to the frequency of the note played as the velocity of the finger hitting the key maps to the loudness.

The number of parameters a performer is able to control simultaneously is limited. Therefore instruments, like the piano, which offer a large polyphony, do not allow as much control of note envelope parameters as monophonic instruments do. Some instruments, like a violin, offer both polyphony and a high degree of continuous control. But the latter is not fully used when played in a polyphonic manner.

Mapping consists of projecting a N -dimensional space of control parameters onto the M -dimensional space of the algorithm's variables, where N is generally larger than M . Even though mapping is not only linked with real-time processing nor with the use of gestural controllers, it is obvious that it has the most acute implications in that context. What parameter change should be associated with a given gestural input? What kind of gestural interface is best suited to control a given algorithm? There is no immediate answer to these questions. It depends on the user's experiences and preferences and on his artistic vision, as mapping usually integrates implicit or explicit rules that define relationships between parameters. We will nevertheless review two important aspects of mapping: assignation and scaling.

12.3.1 Assignation

One can split the various ways of assigning the space of control parameters to the space of the algorithms variables into four categories:

- One-to-one assignation: an example is the tuning of a filter, where a frequency control is used to modify the center frequency of a filter. Another is the positional mapping of the note on a synthesizer keyboard.

- One-to- M assignation: an instance of this is the case when a MIDI velocity message from a keyboard (giving information about how fast a keyboard key was depressed) is used to control the volume of a sound, its brightness (for instance modifying the cut-off frequency of a lowpass filter), its attack time, etc. An interesting example is the constraint-based spatialization approach [PD98], where changing the position of an instrument in space also moves several others depending on previously defined constraints.
- N -to-one assignation: Still using an example involving a keyboard control, a filter cutting frequency might be controlled both by the velocity value, for individual notes, and by a control wheel to increase or decrease the overall brightness. Additionally, it could also be controlled by a foot pedal or by a breath controller.
- N -to- M assignation: We will give the GUI example of the SYTER interpolation screen in Fig. 12.3, where 2-D mouse movements are used to control 16 parameters.

12.3.2 Scaling

Once we have assigned a control input to an algorithm parameter, we still have to decide if and how we will scale the value given by that input to control effectively the chosen parameter. The sound processing algorithms usually use internal variables ranging from -1 to 1. This was the rule with DSP (Digital Signal Processors) chips which use a fixed-point number representation. This is now changing with the availability of cheap and powerful floating-point DSPs and with an increasing number of programs running on the main computer's microprocessor. However, this does not change the fact that a "gain" value used by the algorithm relies on an internal linear representation ranging from 0 to the maximum. As the user perceives it according to a logarithmic scale, expressed in dB, a scaling (in fact a one-to-one mapping) must be applied to transform the values entered by the user in dB to the linear representation needed by the algorithm. The reverse is also true: when displaying an internal volume on a level meter that makes sense to the user, the internal linear value must be transformed into a logarithmic value. We can express this by $y = F(p)$ or $p = F^{-1}(y)$, where p is a perceptive value in dB or in Hz and y is the internal value in the range $-1 < y < 1$. Depending on the type of parameter one needs to control, $F(p)$ and $F^{-1}(y)$ can take different forms. Sometimes an offset b and a gain factor a are useful according to $y = a(p + b)$ and the inverse operation $p = \frac{y}{a} - b$.

Logarithmic scaling. Perception usually follows what is known as Steven's law: we are more sensitive to relative changes, $\frac{\Delta y}{y}$, than to absolute changes of the internal values. Changing the attack time of a sound from 100 to 200 ms is very noticeable, but changing it from 10000 ms to 10100 ms will not be heard. The absolute change is 100 ms in both cases, but there is a 100 percent increase in the first one and only a 1 percent increase in the second one. In assigning a linear scaling to a MIDI fader, which offers only 128 steps from 0 to 127, to an attack time that

you want to go up to one minute, the minimum time increment is $\frac{60000}{127}$ ms = 472 ms. This does not give enough precision in the lower end and wastes precious steps in the upper end. Using a logarithmic scale, one achieves the same relative precision all the way through the run of the fader. This may be expressed by the formula $y = ab^{\frac{p+c}{d}}$ and the inverse operation by $p \log_b \frac{y}{a} - c$. We know for instance that the perception of pitch follows such a scale, with the frequency doubling for each octave made out of 12 half tones. If we then want to map incoming MIDI notes to frequencies, knowing that note number 69 corresponds to a A at 440 Hz, we could use the following values: $a = 440$, $b = 2$, $c = -69$ and $d = 12$. If we want to map a MIDI potentiometer to a gain with 1 dB steps, where MIDI value 127 corresponds to 0 dB and to the internal value of 1, we could use $a = 1$, $b = 10$, $c = -127$ and $d = 20$. The complementary equation can be used to drive a level meter from the internal linear volume value.

Approximations. The computations of log and exp functions might be too computer intensive when the processing is done on fixed point DSPs. Curves might be calculated out of real time, when ranges of the parameters are entered by the user, and put into tables. Those tables are then read in real time, eventually computing the missing points with simple linear interpolation. Another solution, used in TC Electronic effect units [Nie99], is to let the host processor compute off-line the coefficients of a third order polynomial ($y = ax^3 + bx^2 + cx + d$) that best fits the desired scaling. The DSP can then effectively perform the polynomial calculation in real-time. When there is no previous knowledge about what kind of law applies best, one may always provide a system like a table, with the input value on the x axis and the output value on the y axis. The user then draws the most appropriate curve by trial and error.

12.4 GUI Design and Control Strategies

12.4.1 General GUI Issues

One could say that every strategy that proves useful for a particular user in a specific context is valid in that special case. But it is important to remember several general issues that have proven to play an important role in any GUI design [Mei91, pp. 57–62]:

- Visibility: it allows the user to see what he can do with a given tool.
- Transparency: the user does not see what has been intentionally hidden from him (what the computer system is really doing), but he is given a way to visualize the task being performed according to the mental image he has built up.
- Foresee ability: the system performs the task that the user naively expects it to do, building up on his previous knowledge, often with the help of metaphors, like a piano keyboard or a fader.

- Consistency: the interface is foreseeable in every context within the program and from one application to another.
- Integrity: the interface protects the precious data even if the user makes a mistake; there should be a way to cancel or undo certain actions.
- Concision: it is a very important element, both at the control level (short-cuts, pop-up menus, default values, etc.) and at the screen layout level (especially the useful information).
- Screen appearance: the screens have to look good, be clear and well ordered; brightness, color, textures, flickering should be used mostly for their meaning than only for aesthetic reasons.
- Adaptability: the user may, without the need to program, configure the interface to suit his needs and his level of knowledge.
- Guiding: every user sometimes needs answers to certain questions such as “how can I quit this program?”; if the interface is not self-explanatory, an on-line manual or contextual help may be useful.

12.4.2 A Small Case Study

Let us imagine that a sound engineer, familiar with analog compressors, is presented with a neat GUI comprising five potentiometers clearly labeled gain, threshold, ratio, attack and release (see Fig. 12.2). He may move the potentiometers with the mouse and values indicated are given in the proper units (dB, dB, -, ms, ms). It is obvious that most of the issues raised before are answered: visibility, transparency, foreseeability, concision and screen appearance. Integrity is not really an issue here, though one may add ways to name and save parameter configurations for later recall. Consistency within the application is already evident, but if a noise gate or an expander or a bunch of other audio effects are proposed with the same type of interface and self-explanatory names, consistency between applications will also be met. The screen appearance in Fig. 12.2 is very clear and, as an additional help, the transfer function is also shown graphically.¹ A certain level of adaptability is achieved by allowing the user to modify the knee type and to activate a side-chain. If the potentiometers cannot be moved during the computation of the sound result, a score function that enables the user to define the evolution of the parameter's values over time before computation would be a welcome addition.

Another important category of visual interfaces relies on a representation where time is mapped to the x axis and frequency to the y axis. The first of this kind, with a discrete frequency axis, was the UPIC [Loh86]. It used the time-frequency representation only as a control interface to literally “draw” a composition. Closer to the sonogram, AudioSculpt is a program where several tools allow the user to modify a graphical representation of the signal over time, obtained by FFT analysis,

¹<http://www.digidesign.com>

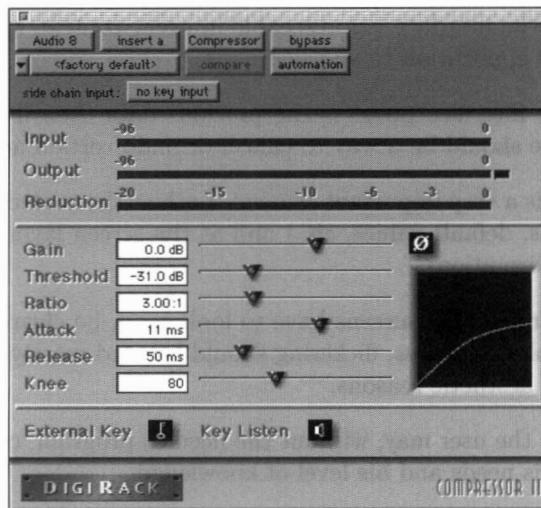


Figure 12.2 Plug-in compressor for a screen-based mixing program.

before proceeding to the inverse FFT and to synthesize the transformed sound. Some important issues about that category of interfaces are discussed in [Arf99].

12.4.3 Specific Real-time Control Issues

If the sound is computed in real time, the ease of use of this audio effect will be improved as the user will be able to fine-tune parameter values while hearing the resulting sound change accordingly. He will be given one of the feedback channels described in section 12.2.

Modern MIDI sequencers and Direct-to-Disk recording programs² running on multiple platforms, are good examples of the implementation of these principle and third party plug-ins often following most of the same rules. These environments also allow the user to customize his virtual studio without too much hassle. But even then something important is missing: in the real world the user could have twisted two of those potentiometers simultaneously or jumped from one to the other almost instantaneously, using his kinesthetic knowledge of their position. This is one of the big differences between a musical instrument and a screen-based GUI. One writes with one hand, but one usually plays music with both hands, often quickly changing hand positions. Standard computer interfaces only acknowledge the use of two hands to enter text on a typewriter-style keyboard. We see that, even though translating the studio metaphor to the GUI makes a digital audio effect easy to understand and to operate, it does not translate the playing modes.

Simultaneous access to all parameters with the help of MIDI faders would restore

²<http://www.digidesign.com>, <http://www.emagic.de>, <http://www.steinberg.net>

the highly praised studio habits and allow the user to make faster adjustments. Also, adding automation would expand the possibilities of the digital audio effect beyond those of its analog counterpart. This example shows that new issues have to be addressed on top of the general computer interface issues, because the way an effect sounds depends heavily on the control given to the user. Different ways to access the parameters lead the user into different paths and experience has shown that even minor changes to the user interface can dramatically raise or lower the creative use of a digital audio effect.

One should also point out that it is not always a good idea to stick to the studio metaphor, as an overly conservative approach might unnecessarily limit the power of an algorithm. As an example of this, we have seen many composers starting to use a band-pass filter plug-in³ only when a 2-D control window allows them to change both the center frequency and the bandwidth in one single gesture, by moving the cursor with the mouse along the x and y axis. It suddenly became more than just a filter though the underlying algorithm had not changed at all.

12.4.4 GUI Mapping Issues

This latter example introduces the important notion of mapping that is discussed in section 12.3. Mapping usually consists in projecting a space of control parameters into the space of the algorithm's variables. Unlike the bandpass filter example given above, those two spaces have often different dimensions. An interesting graphical solution was proposed in 1984 by Daniel Teruggi (SYTER SYstème TEmps Réel, [All84, All85, Dut91, Ges98, Ter91, Ter94, Ter98]) which was a very user-friendly real-time digital signal processing workstation built on their previous experience [AM81]. The INTERPOL window (see Fig. 12.3) allowed the user to size and position numbered circles in a 2-D plane. Each circle was assigned a set of up to 16 parameters controlling the ongoing sound processing. Whenever the user clicked inside one of those circles, the corresponding set of parameters was instantly recalled and they could be used as presets. But it was also a metaphor for a gravitational system, where larger objects have a further ranging influence. By moving the cursor along the screen, one was able to continuously interpolate between the selected parameters of the presets. When reaching an interesting sounding point, the user could also create a new circle with that very set of parameters, changing the whole distribution of values across the 2-D plane at the same time. It is a powerful empirical way to define, by trial and error, the domain of a digital audio effect one wishes to explore.

These ideas have been further developed for 3-D space [TTL97, TT98] on the NeXT-ISPW (IRCAM Signal Processing Workstation). Figure 12.4 shows several spheres associated to sets of parameters. Additionally, the user could record trajectories in the space and recall them on the fly. They could be scaled, played at various speeds and reversed. As the same interface was designed to spatialize sounds, it became an interesting tool to experiment with spatio-timbral correla-

³<http://www.ina.fr/grm/>

tions. These examples show some form of adaptability of the interface: the control space is defined by the user.

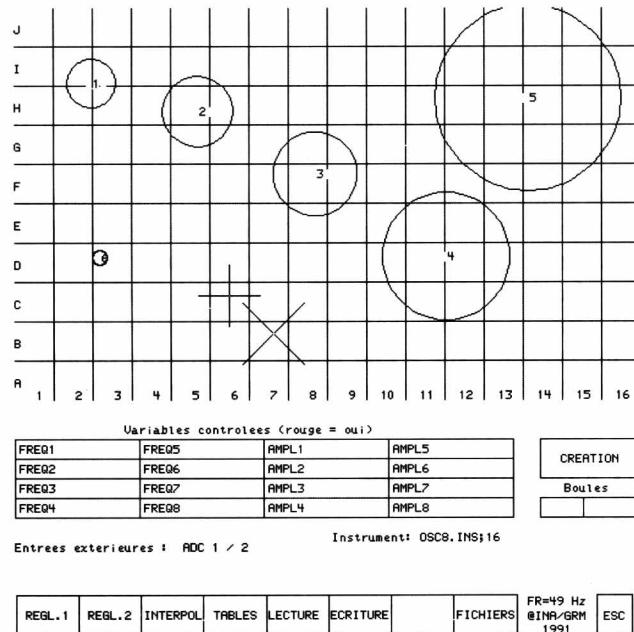


Figure 12.3 “Interpol” control screen of SYTER.

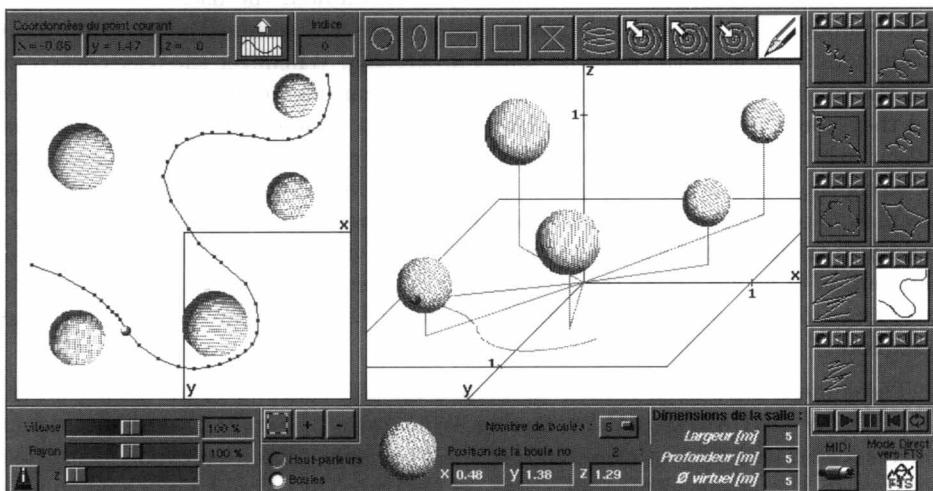


Figure 12.4 3-D interpolation control screen.

12.4.5 GUI Programming Languages

Going one step further in adaptability, we find the graphical programming environment. They are usually based on the modular analogue synthesizer or on the studio metaphor. A library of modules may be combined to create complex patches including both control paths (usually asynchronous, depending on the user actions) and audio paths (usually at the sampling frequency or downsampled by an integer factor). The oldest program of this kind is MAX™, originally developed by Miller Puckette at IRCAM. Signal processing modules were added later [Puc91a, Puc91b]. Several offsprings [Zic97] have followed (jMAX, PD and MAX/MSP), running on several computer platforms. Figure 12.5 shows a simple patch performing additive synthesis with 4 sine oscillators. One can see the signal processing modules, ending with a “~”, the number boxes for entering values, toggle buttons, a preset box and a virtual oscilloscope. Tables, sliders, multisiders, meters, drawing windows and envelope editors are also available, as well as many data control and signal processing modules.

This kind of environment takes much more time to master, but the user has complete freedom in designing the audio effects and controls he wants. It is also an open environment where one can program new modules in C. This has led to a large offer of third party modules performing many different functions: data control, signal processing, visualization and drivers for various hardware.

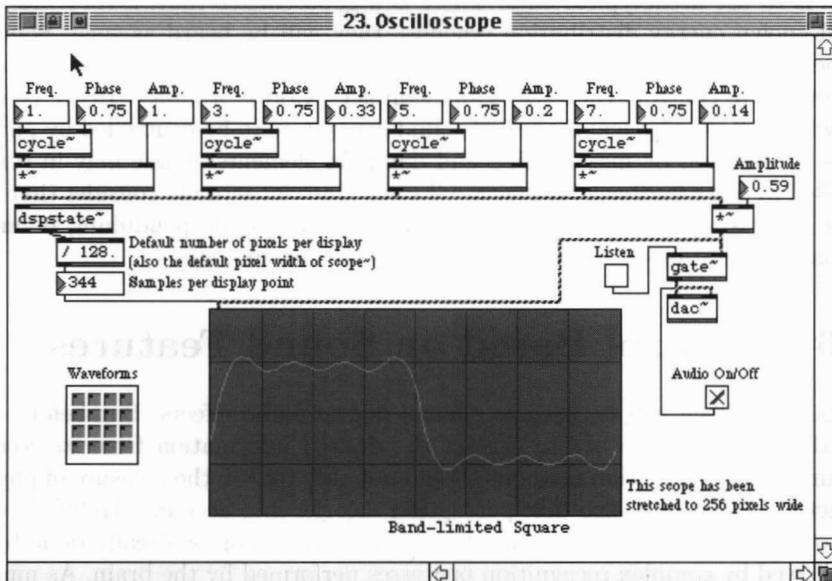


Figure 12.5 A simple patch.

12.5 Algorithmic Control

12.5.1 Abstract Models

Many different abstract models have been proposed for sound synthesis (generating and distorting sound waveforms) and control. We will only look at the later ones. The goal is generally to control a multitude of sound events globally. Xenakis used stochastic models early on in [Xen92, m-Xen95] to evoke sound environments that marked his youth. Applications of the same strategies have been used by many authors to control granular synthesis, where a multiple of independent grains have to be generated. A refinement is the use of tendency masks that describe a time evolution of minimum and maximum values that constrain the stochastic output. Cellular Automata [Bey89] have been used to control the same algorithms in an attempt to recreate an organic evolution over time. Genetic Algorithms are less straightforward in that the time evolution is not obvious, but they seem to be good candidates to control the morphing between two different states. Any mathematical model could in fact be used and the serial approach used by the school of Vienna composers may be considered one of them. The only question should be whether they control the sound processing algorithm in an interesting way.

12.5.2 Physical Models

One way to answer the concerns about the auditory perception of those underlying models is to use archetypical ones. There is a good chance that, if they are based on well-known energy distribution schemes, they will be heard as such. Some approaches use models of bouncing balls, fortune wheels, etc. to generate streams of MIDI events to control synthesizers and samplers. Cordis Anima [CLF93] is a program which allows physical models of sound generators to be controlled by physical models represented by mass, spring and damping elements. These may in turn be controlled by force feedback interfaces [CLF90]. One can actually consider that some uses of stochastic functions fall into this category and so do pendulum, bouncing, rotation, acceleration-deceleration, aggregation-dispersion, etc.

12.6 Control Based on Sound Features

The sound in itself can be used to control digital audio effects. But before using it for that purpose, one has to extract the desired information from its complex structure. These extraction methods should not only rely on the measure of physical parameters. They must also take perception theories into account [Tro99]. If we go back to Fig. 12.1, we see that some kind of pre-processing is already done by the ear, followed by complex recognition processes performed by the brain. As much as scaling is important for visualizing internal parameters of algorithms, sound data representation has to be based on the concept of Just Noticeable Differences (JNDs). Models of hearing have been established and they can help us extract features which have a perceptual meaning.

12.6.1 Feature Extraction

Many feature extraction algorithms have been discussed in the previous chapters. The most common ones in practical use are:

- *Pitch tracking*: allows the computation of the fundamental frequency of a monophonic sound input. The first methods were developed for speech processing [RS78]. The voice is a particularly difficult signal, as the pitch might show rapid and constant changes [Puc95]. On top of that, the onset of a note can have an instantaneous pitch several half-tones away from the note on which it stabilizes. Then there is the problem of vibrato. Other instruments like piano, flute or clarinet are easier to follow, as the first harmonic is present and one can rely on a tempered scale. Some systems are able to perform polyphonic pitch extraction in certain contexts [PAZ98], but the problem is far from being solved for all situations, specially when it has to work in real time. Because of the difficulties of recognizing chords after identifying individual notes, certain researchers choose pattern matching techniques on a semi-tone intensity map derived from spectrum analysis, with a database of chord-type templates [Fuj99].
- *Amplitude tracking*: also called envelope follower, is a program that extracts the power of an audio signal, usually computing its rms value. Extra parameters allow adjustment of raise and fall times. This information may be used to trigger sound effects on and off by defining absolute or relative thresholds.
- *Centroid tracking*: this information gives the evolution of the gravity center of the spectrum obtained from the FFT analysis (see section 9.4.2). The FFT is computed for each frame of about 50 ms duration, with a Hanning window and with a 50% overlap. For the typical sampling rate of 44100 Hz we choose $N = 2048$. The frame duration is hence 46 ms and the hop size is 1024 samples or 23 ms.
- *Voice/silence and voiced/unvoiced tracking*: are features originally used in vocoders to switch the carrier on or off and to decide whether to use the internal glottal pulse generator or the noise generator for consonants. The latter often has a Boolean output, but there may also be some kind of percentage of noisiness for voiced consonants. Detection usually consists of counting the number of zero-crossings of the audio signal in various frequency bands.
- *Partial tracking*: each of the partials (not necessarily harmonic) of an audio signal is extracted, generally by FFT-based methods.
- *Rhythm tracking*: a less common, but musically useful feature is the rhythmical structure and tempo. Tracking it is a complex task, often requiring the use of artificial intelligence techniques.

12.6.2 Examples of Controlling Digital Audio Effects

Several examples have been shown in the previous chapters, where features extracted from a sound control the processing of the same sound: dynamic processing, denoising, etc. But one can define many different processing tools from the same basic building blocks. For instance, a voice/silence detector can control a time stretching program in order to avoid time stretching of silence parts [Tru90].

Another category of control involves the use of features extracted from one sound to control another one. Modulation, vocoding or more generally cross-synthesis and hybridization have been described in previous chapters. In a concert, partials extracted from groups of instruments can be classified according to perception theories, individually modified and resynthesized or used to control other sounds [m-Fin95, TSS82, TDF95].

In the same spirit, tracked parameters may also be used to control music in interactive dance performances. Figure 12.6 shows how a MSP module called “rms~” generates an output signal from the RMS value of an incoming audio signal, using minimum and maximum values as well as low and high thresholds with associated slopes (performing a combination of a downwards expander and a compressor in this example). With associated time-parameters such as attack, decay and hold, the module will react on specific energy levels and articulations. The right part of the same picture shows a schematic view of four such modules, each with its own set of parameters, processing the sound from a single contact microphone. This kind of setting was used in the dance performance “In Between” [m-Noi00] to control the playback volumes of several sound files and the volume of the sounds picked up by the microphone itself. Adding effects and spatialization over a group of 8 loudspeakers, the dancer can control a complex sound environment simply by touching a dedicated surface.

Another important use of feature extraction is score following [PL92, Puc95, PAZ98], where the computer tries to follow the performer, only by “listening” to the sound he produces. It is done by matching the extracted pitches with a previously entered score. This frees the performer from having to follow the computer and allows him an increased level of expressivity.

12.7 Gestural Interfaces

Gestural interfaces complement all the other control methods already described in this chapter. We pointed out earlier that even with an ideal GUI, something is missing when it comes down to transforming a digital audio effect into a playable instrument. The various gestural interfaces available today bridge this gap. The idea of an instrument has considerably evolved since the times where only acoustical ones existed. One does not have to excite a vibrating body anymore in order to make sound, but the kinesthetic and haptic aspects remain important. A very large variety of interfaces [Par97, Wan97] have been built. Several authors have made classifications of categories of gestures [CLF90, Mul98]. Cadoz divides them into

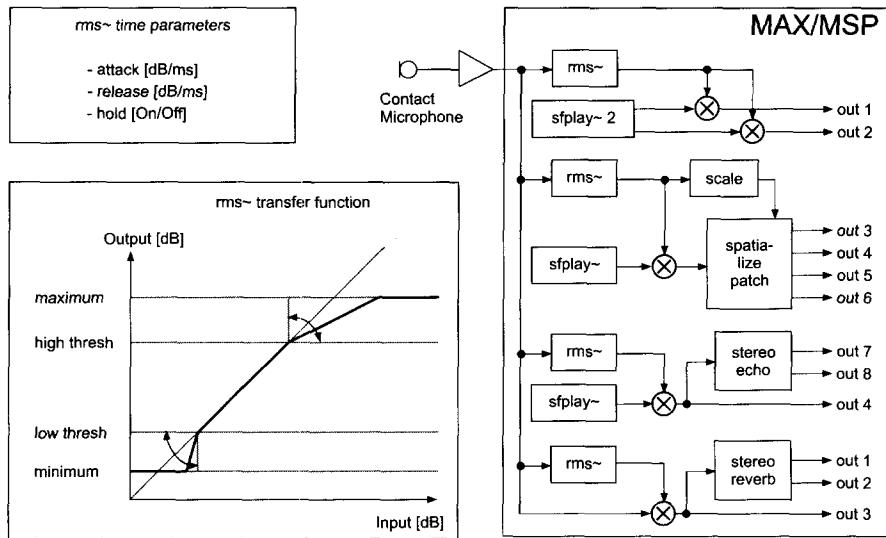


Figure 12.6 rms~ detection module with its transfer function and a schematic example for an interactive dance performance.

excitation, modulation and selecting gestures. Often the question arises of which is the best transducer for a specific musical function [WVIR00].

We will divide gestural controllers into four main categories: gestural interfaces played by touching or holding the instrument, interfaces with haptic feedback, interfaces worn on the body and interfaces that may be played without any physical contact. First, we will have a look at the advent of MIDI.

12.7.1 MIDI Standard

The MIDI standard specifies both the hardware interface and the data transmission protocol. The hardware MIDI serial interface operates at 31.25 kbauds, asynchronous, with a start bit, 8 data bits and a stop bit. The interface uses a 5 mA current loop with optoelectronic detectors at the receiver end to avoid risks of ground loops. The protocol is a definition of status bytes, defining the action, that may be followed by one or more data bytes, defining the values for that action [MIDI]. The very existence of such a standard MIDI interface has done a lot for the proliferation of gestural interfaces, as any of them may easily be connected to almost every digital audio device or program. The range of commercially available MIDI controllers is still expanding. Keyboards, guitar controllers, breath controllers, percussion modules, drum kits and fader boxes have been joined by more exotic controllers from big manufacturers, like the Roland D-Beam™ infrared system, or the Korg Kaoss Pad™ tactile 2-D surface, showing a growing need for continuous controllers. More specific controllers like the Matthews Radio baton, the EMS SoundBeam or the

Theremin [Smi00] can be bought in MIDIfied versions.

NoTAM MIDIconverter, STEIM SensorLab™, Infusion Systems I-Cube™ or IRCAM AtoMIC™ are some of those more universal systems that translate data from a whole range of sensor systems to MIDI, allowing artists to customize their interfaces. They are perfect for people who make interactive performances or sound installations as they accept a large range of sensors and can be used without a lot of technical knowledge. Using some cheap and easily available microprocessor development tools, every sensor may be transformed into a MIDI controller and there are many possibilities: pressure, flexion, position, speed, acceleration, and conductivity sensors, hall-effect, magnetic, electrostatic and capacitive detectors, light cells, anemometers, etc.

The vast majority of gestural controllers have adopted the MIDI standard. Most of the computer-based interfaces are able to exchange data with other MIDI programs, being thereby able to send out MIDI information when equipped with the right hardware and software.

12.7.2 Playing by Touching and Holding the Instrument

Keyboards

The piano-like keyboard has without doubts been the most used controller in the history of electronic instruments and digital synthesizers. Many MIDI status bytes messages have been devoted to the needs of keyboard players: besides triggering notes (Note On/Note Off), keyboards usually send information about the speed at which a key has been depressed (Note On Velocity), and often information about the speed at which it has been released (Note Off Velocity). Information is sometimes sent about the pressure the player exerts after the note has been depressed, either globally (Aftertouch) or independently for each key (Polyphonic Aftertouch). A wheel is usually available for controlling the frequency (PitchBend) and another for the control of vibrato or tremolo (Modulation, part of a list of 127 available Continuous Controllers).

Besides traditional keyboard design, some atypical ones offer a different layout, allowing playing of microtonal music. Moog proposed the Multiply-Touch-Sensitive Keyboards (MTS) [MR90], sensing the (x, y) position of the finger on each key with resistive films, the up-down position of the key (z axis) with a capacitive system and the aftertouch with a resistive film. In [FA00] a continuous monitoring of the keys vertical position is performed by an optical system. In order to resolve the problem of the high data rate, which cannot be transmitted through MIDI, data is embedded in standard SPDIF or ADAT audio streams. Compared to aftertouch, which only begins sending data once the key is completely depressed, these two systems provide information during the whole run of the key. With Stahnke's Bösendorfer 290 SE prototype [MR90], the Yamaha Diskclavier is a category in itself, it consists of an acoustic grand piano fitted with sensors that monitor the depressed keys and send the related information to MIDI. Actuators under each key make it in turn possible to play the piano by sending MIDI messages. Jean-Claude Risset developed MAX

objects to both monitor and play the piano [RD96] and several of his interactive compositions made use of them [m-Ris]. Some force-feedback techniques have also been implemented and will be described in a forthcoming section.

Percussion Interfaces

The first percussion interfaces were acoustic drums fitted with pickups. An envelope follower with a threshold was used to define a trigger level and the intensity of the picked-up sound controlled various synthesis parameters such as attack time, filter cut-off frequency, amplitude, etc. Specific drum-like interfaces, without resonating bodies, were later introduced and offered by all main instrument manufacturers. The next step was the introduction of control surfaces that not only reacted to the force with which they were hit, but also to the position and to the pressure after the hit. Buchla's Thunder is a good example. The fact that MIDI data may easily be mapped to various parameters makes it a very versatile instrument. The Korg Kaoss Pad™ tactile 2-D surface may also be used as a percussion instrument, reacting to the position being hit.

String Instruments

String instruments may also be transformed into MIDI controllers. The most commonly used technique is to fit the instrument with multiphonic magnetic pickups (one for each string) followed by a pitch tracking device that transforms the sensed frequencies into MIDI notes and pitchbend information. The difficulty lies in detecting fast enough rapid changes in pitch. Some interfaces use a different approach: the stick is covered by a sensitive surface that senses the finger's positions and the pickups are merely used to measure the amplitude of the string's vibrations. In some cases, the strings completely disappeared leaving only tactile sensors and switches to track the guitarist's gestures. The MIDI throughput is still a limitation and faster interfaces are needed to transmit the whole subtlety of a guitarist's playing. Another solution is to send the audio signal from the pickups directly to the computer and have the pitch detection algorithm running on the computer where the controlled sound is generated. Several techniques have also been proposed to track the bow position relative to the instrument and its pressure on the strings.

Wind Instruments

Two different approaches prevail: one is to replace the acoustic instrument by an interface measuring breath force and/or bite pressure on the reed and by providing sensors that track the fingering of the simulated instrument. The other approach is to analyze the sound of a real acoustic instrument, and extract the gesture from the sound. Both systems may be combined by fitting sensors on an acoustic wind instrument.

Hyperinstruments and Gesture Extraction

The concept, defined by Tod Machover, consists of expanding the playing modes of traditional acoustic instruments. Techniques usually combine feature extraction from audio data as well as specific sensors fitted to the instrument [MC89, m-Mac]. Other researchers focus on the extraction of gesture parameters detected solely from the sound of an acoustical instrument allowing the performer to play his instrument normally, without any added sensor [Wes79]. Starting from the knowledge of the playing modes, the features of sound generation, and constraints of a given instrument, it is somehow possible to guess what kind of gesture has produced the analyzed sound. This educated guess may then be used to control parameters of a digital audio effect. The principles are similar to some techniques used in speech recognition, where formant extraction is performed by looking at the possible configurations of the vocal tract knowing their constraints. This could improve the concept of hyperinstruments one step further.

Batons

The first batons were built with the analogy of the conductor in mind [Bou90]. Their goal was to have the computer-generated sounds follow the conductor just like the other performers in the orchestra. It should therefore be able to track beats, extract the tempo, detect accents, nuances. This comes down to a type of pattern recognition system which has to recognize and decode the complex gestures of a conductor. Different systems were used. The MIDI Baton by David Kean [KG89, KW91] used a conductive contact ball attached with a spring wire inside the conductive tube serving as baton. Whenever the direction of the baton changes suddenly, an electric contact between the ball and the inner tube is made and detected. The tempo is tracked and converted into MIDI clock messages. The AirDrum, manufactured by Palmtree Instruments, has a 2-D accelerometer and a rotation detection. As an extension, [IT99] has proposed not only to track the tip of the baton, but also body and hand movements with the help of magnetic motion trackers. The data is then processed with the help of neural networks.

Even though the Matthews and Boie Radio baton was developed in the late 1980s mainly from a conductor's perspective, its design steadily improved and is now mostly used as an instrument [BM97]. It features two sticks, one for each hand, with small coil antennas at the end, each transmitting on a separate frequency of around 50 kHz. The body of the drum hides an array of five flat receiving antennas, two on the left and right sides, two on the upper and lower sides and one in the middle. An (x, y, z) position is computed, for each stick, from the five intensity levels. The instrument may be used as a triggering device by setting a vertical threshold, or serve as a double 3-D controller. A modified version of the system could also theoretically be used on stage, with large flat antennas under a dance floor to track moving dancers. Often, one stick is dedicated to triggering events depending on the position, and the second stick is used to move within a timber space. Arfib [AD00] uses the radio baton to control a digital version of the intriguing photosonic instrument. The musical piece "The Nagual" [m-Mai97] is designed as

a duet between a percussionist playing a set of metal objects and a radio-drum player controlling computer-generated sounds. The computer produces sounds that are derived by filtering white noise. The role of the computer-performer is to shape the sounds in such a way that they tend to be similar to those of the physical objects and to complement the musical gestures performed by the percussionist. To reach this goal, the computer-performer uses a radio drum that simultaneously drives the tuning frequencies and the quality factors as well as the amplitudes of two banks of eight filters each.

Another successful and commercially available baton controller is The Buchla Lightning II. Each of the two batons is fitted with a modulated infrared LED and tracked with a photodiode array in the receiving station, up to six meters away. The MIT MediaLab Digital Baton [MP97] detects the 2-D position of the edge of a baton, also fitted with an infrared photodiode, with the help of an infrared camera. In order to compensate for the delay introduced by any camera-based system (only 25 or 30 frames per second), an additional three-axis accelerometer helps tracking fast gestures. Five force-sensitive resistors included in the baton also measure hand and finger pressure. It was used in the “Brain Opera” [m-Mac].

Flat Tracking Devices

A driver for standard Wacom tablets is available as an external MAX module [WWF97]. It allows the use of those tablets as input devices to simulate the playing modes of existing instruments or navigation in a timberspace. A Wacom tablet may for instance be used to control bowed string instruments [SDWR00] in Max/MSP. The proposed mapping is as follows: y position → bow position, derivative of x position → bow velocity, z position → bow pressure, tilt angle in the x axis → string played and tilt angle in the y axis → amount of bow hair.

Several proposals have been made to measure the (x, y) position continuously and pressure independently for each finger. The Continuum [HAS92] is a polyphonic controller that tracks independent (x, y) position and pressure. Several prototypes were made, using various technologies. Tactex recently proposed the MTC Express™, a commercial product based on a grid of interleaved optic fibers enclosed in a special fabric, that is able to track position and pressure of up to five fingers. Though there are only 72 crossing points, centroid computation permits precise position detection, up to 100 dpi and 256 levels of pressure. At this moment, the sensing surface is still small (15 by 10 cm), but larger models should follow.

Other Interfaces

So many original hands-on controllers have been designed that we can only cite a few, chosen for their diversity.

- A non-contact optical tracking device is the VideoHarp [RM90] which is a light sensitive device that allows the tracking of fingers in a flat, hollow, rectangular frame.

- “The Meta-Instrument” [Lau98] is a man-machine interface for control of algorithms for sound synthesis or multimedia applications in real time. It consists of two hand and forearm controllers and two foot pedals.
- The aXi0 MIDI controller [Car94] is a long stick-like instrument that stands on the ground and rests over the left shoulder of the performer. The right hand controls a chord keyboard while the left hand rests on a palmrest, equipped with several switches and a touch strip, sitting on top of a 3 degrees of freedom joystick.
- The Sentograph [VU95] is a kind of push-button sensitive to x , y and z position. It is not a very precise device in the sense that it is quite difficult to control each of those three freedom degrees independently. But the idea is to capture a global gesture, intuitively and emotionally, rather than very precisely. It is part of the wider project for the development of a sort of control cockpit with a topography that may be constantly re-adjusted depending on the controlling needs [VUK96, UV99].
- The Gmebaphone [CGL98] is an interface designed to control the spatialization of tape music during a concert. Touch-sensitive faders with visual position feedback are used to send the music to groups of loudspeakers placed on stage and around the audience.
- A very original approach uses the individual magnetic resonating frequencies of objects to locate them in space. Up to 30 wireless magnetically coupled resonant tags may be tracked in the same control space without affecting each other [HP99]. Such tags may be attached to each finger of a performer, or included in various objects that can be moved on a surface, a bit like pieces on a chess game. The system returns the center frequency, resonance width and integrating coupling amplitude for each tag. The latter provides an indication of the tag’s distance from the reader and their mutual orientation, enabling continuous non-contact control. Tactile parameters may also be acquired by making the resonance frequency parametric with pressure.

12.7.3 Force-feedback Interfaces

Recognizing the importance force-feedback plays when performing an acoustic instrument, some researchers have investigated ways to transmit haptic information to the performer of a digital instrument. The main aim is to offer a more natural style of interaction between the player and his instrument. Several researchers have focused on trying to recreate the feeling of playing a real piano by simulating the changes in the key’s resistance along its vertical path. Gillespie [Gil94] proposed use of a finite state machine as the couplings between the key, the keybed and the hammer changes dynamically when depressing one key. The vBow [Nic00] provides haptic feedback for the bow of an electronic violin. Others have tried to expand the concept to totally new instruments. Long-term research has been led at ACROE in the development of missing tools. One problem was the lack of suitable motors

able to generate enough force-feedback while remaining of reasonable size [CLF90]. The other problem is to find a method to model the link between the interface and the sound generating program. The answer lies in a total integration between both, thanks to physical models of sound generation.

An interesting interface was created in an attempt to let blind users access window-based computers. The Pantograph [Ram95a, Ram95b] is a sort of force-feedback mouse with which one can feel the window's limits and get directed to locations where to drop files. Though not specifically designed for musical use, one can easily imagine musical applications, like finding relations between resistance and the concept of musical tension, or having the interface returning spontaneously back to an equilibrium position. In the meantime a force-feedback mouse is on the market place. Further haptic interfaces for musical applications can be found in [Bon94, Cha93, Chu96, RH00].

12.7.4 Interfaces Worn on the Body

Rather than using an instrument, another approach consists of measuring body parameters directly, like body temperature, skin electrical resistance, eye activity through EOG (Electrooculogram), brain activity through EEG (Electroencephalogram), muscle potentials or EMG (Electromyogram), heart activity with ECG (Electrocardiogram). Many bioelectric musical recordings and performances were produced in the 1960s and the 1970s, under the generic name of biofeedback [Ros90]. The BioMuse [KL90] measures bodily electrical activity (EEG, EOG and EMG) and transforms them into MIDI data after filtering, frequency analysis, signal recognition and signal comparison. The BioMuse comes with a series of electrodes mounted on velcro bands, a small battery-powered patchbox that may be worn on a belt, and the signal processing unit that receives the amplified signals from the patchbox, performs the processing and sends out analog and MIDI data. The MiniBioMuse [Nag98] is a much smaller and cheaper alternative, but with less inputs.

Other systems measure relative or absolute positions of body parts. The best known device of this kind is the "Data Glove", which measures finger flexions and hand position. Laetitia Sonami has been performing for years with such a glove designed at STEIM and Sonology. It adds an ultrasound sensor to measure relative distance between both hands and Hall effect sensors on the finger tips to precisely measure small distances between the fingers and the thumb. The Exos Dexterous Hand Master has been used by Tod Machover and instead of relying on flexion sensors, it directly measures angles of each finger's phalanx with much better precision, but with the disadvantage of a cumbersome mechanical add-on. Michel Waisvisz was certainly the pioneer in using hand controllers. His system, the "Hands" [Kre90], was used in numerous performances. It differs from the others in that it is not a measure of finger flexions. It is a rather complex instrument made of two parts, attached to each hand, that he plays by pushing buttons. Special software has been developed for the instrument. A real-time sample recorder and player with numerous parameters are dynamically assigned to the Hands. We find all categories of gestures: some switches serve as selectors to change the context, others to trigger

sounds, whilst the latter modulate them.

Feet also convey useful information about dancers' movements. The Dancing Shoes [PHH99] is a pair of sneakers, each sensing 16 different tactile and free-gesture parameters. They rely on a mix of various technologies. Three force-sensitive resistors are located in the forward sole, a piezoelectric foil measures the dynamic pressure at the heel and two back-to-back bend sensors measure the sole's bi-directional bend. A vertical gyroscope responds to twists and spins. A 2-axis low-G accelerometer picks up tilt and general foot dynamics. A 3-axis high-G piezoelectric accelerometer gives directional response to rapid kicks and jumps. A 3-axis magnetometer gives orientation with respect to the local earth's magnetic field. On top of this, a 40 kHz sonar receives pings from up to four ultrasound sources that may be located at different positions around the stage to measure absolute position. All sensor values are sent, independently for each shoe, 50 times per second with low-power transmitters. A special C++ MIDI mapping library was written to deal with this high amount of information.

Complete data suits have also been made for immersive virtual reality environments, but do not seem to have been used for musical control, possibly because of their price. The wireless DIEM Digital Dance System offers an alternative [SJ98, Sie99]. It allows up to 14 analog sensors to be put on a dancer or actor who wears the interface and transmitter on a belt and can move freely while the receiver converts those signals into MIDI format. In "Movement Study" [m-Sie97], the dancer wears flexion sensors on her ankles, knees, elbows and index fingers. Instead of being fitted in a suit, a solution that was tried without success was that as the suit moved relative to the body joints with the dancer's movements, the sensors are directly attached to the body with adhesive tape. Laurie Anderson used a much simpler technology in her drum suit: piezoelectric sensors detected when they were being hit, as in most MIDI drum kits.

The NoTAM Control Suit is a MIDI controller suit for use in real-time performance of computer music. The suit has eight strips of semi-conducting plastic material mounted on the chest and arms, and 16 contacts on the back of the hands, in the collar and at the hips. Contacts on the finger tips transfer voltage to these sensors. The plastic strips produce analog signals depending on where they are touched, whereas the 16 contacts are simple on/off switches. NoTAMs own MIDI-converter is mounted on the belt and converts the control signals to MIDI. The piece "Yo" by Rolf Wallin uses the control suit connected to an IMW (IRCAM Musical Workstation) which is programmed with a number of algorithms for granulation and filtering of sampled vocal sounds.

12.7.5 Controllers without Physical Contact

Going one step further, the position of the body might be used without the need for the performer to wear any special devices. These are obviously the best solutions for sound installations but they also offer a greater freedom for dancers.

The "Theremin" was invented in 1919 by a Russian physicist named Leon Theremin [Smi00], originally designed as an alarm device. Two antennas are connected

to a sound producing electrical circuit. One antenna signal controls the frequency of an oscillator, and the other antenna signal controls the amplitude envelope. As a hand approaches the vertical antenna, the pitch gets higher and approaching the horizontal antenna reduces the amplitude. As there is no physical contact with the instrument, playing the theremin requires precise skills. Theremins have been built by several companies over the past decades and every decade delivers new successors based on innovative technologies.

The “Gesture Wall” [PG97] uses electric field sensors to measure the position and movement of the player’s hands and body in front of a projection screen. The projected video and musical sounds are changed accordingly. The performer stands on a plate, which is applied to a radio frequency signal. This signal couples through the performer’s shoes and is broadcast through the body to a set of four pickup antennas located around the screen. The antenna signals change with the distance of the performer from the respective antenna.

The EMS SoundbeamTM uses ultrasound in a reflective way: pulses of a frequency close to 50 kHz are regularly sent through an electrostatic transducer which also serves as a receiver. When an object or a human body is standing in the emission cone, part of the pulse is reflected which is in turn detected by the transducer. As, for given temperature and humidity values, the speed of sound c is known, the distance D is proportional to the delay Δt between the time at which the pulse was sent and the time it is received: $D = \frac{1}{2}c\Delta t$. This distance information as well as the presence or absence of reflection can then be mapped to MIDI messages following the definition of zones, either to trigger notes or to modulate sounds. A more precise version has been built to monitor almost imperceptible movements of dancers [Tod00] within the ultrasound beam. It is used, amongst other things, to control the buffer pointer of a real-time granulation algorithm [m-Noi00].

D-Beam Twin Tower [Tar97, TMS98] expands this idea by fitting four IR receivers around one transmitter, thereby not only measuring the position, but also the shape and angle of the hand.

Systems like STEIM BigEye, David Rokeby Very Nervous System [Win98], or EyesWeb (shown in Fig. 12.7) can directly process an incoming video signal of a performer, a dancer or an actor on stage. An interesting development of the latter is the extraction of barycenters and of expressivity features [CRT99, CCP00, CCRV00].

Litefoot [GF98] is a 1.8 m square surface, 10 centimeters high, with a surface of plywood recessed with holes to accommodate a matrix of 44×44 (1936) optical sensors. It can track feet in two modes, equally responsive. In the reflective mode, the footsteps are detected by the proximity of an object causing a reflection of light back to the sensor that emitted it. It works best when the dancer’s shoes have reflective soles. In the shadow mode, the floor is flooded with light and the footsteps stop that light from entering the sensors.

The principle used in the MIT LaserWall [PHSR00] is the scanning laser range-finder, whereby a laser beam, modulated at $f_M = 25$ MHz, is detected by an avalanche IR photodiode after reflection from the performer’s hands. The received

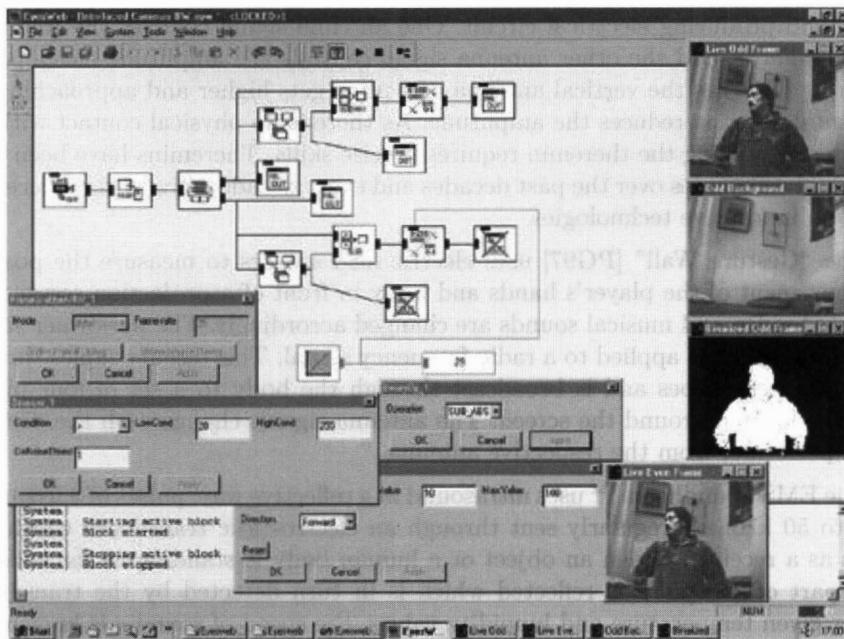


Figure 12.7 The EyesWeb graphical programming interfaces.

signal is multiplied by the modulating waveform in phase and in quadrature, giving two amplitudes from which the phase α can be computed. This phase is directly proportional to the distance between the laser and the reflecting object: $R = \frac{c}{2f_M} \frac{\alpha}{2\pi}$, where c is the speed of light and where α is expressed in rad. As the laser scans the space, which is done by a rotating mirror of known angular position Θ over time, the system gives the polar coordinates (R, Θ) of every reflecting object in the detection space. Microwave motion detectors are added to detect the position of the performer when approaching the wall, before he enters the laser space.

12.8 Conclusion

Being the only link between the user and the algorithm, the control layer is essential to unleash the power of any digital audio effect. We have seen throughout this chapter, with many examples, that there is no single answer to the problem of mapping. At the end, it all comes down to finding a musically useful correspondence between the various accesses given to the user and the parameters of the digital audio effect. As mapping usually creates implicit or explicit rules constraining the simultaneous variations of several parameters, it obviously becomes an artistic choice. And it is no wonder that certain tools are favored in some aesthetic circles. The proliferation of certain algorithms under various disguises is one answer to the different needs.

Allowing the user to choose or to customize his control environment, exploiting fruitfully the flexibility of a virtual architecture, is a better one. We have seen that many concurrent processes may be part of a complete control structure. In fact, every control strategy includes these elements in various degrees:

- GUIs offer various user/performer access and information.
- Algorithmic control allows global control over a large number of parameters.
- Feature extraction maps specific aspects of a complex sound structure.
- Gestural interfaces allow control of gestures ranging from the traditional instrumental playing modes to dance movements, with a steadily growing number of controllers.

It is obviously impossible for the programmer of a DAFx algorithm to spend time providing those many different control layers. Fortunately, it is not needed. With increasingly standardized real-time interfaces and protocols (MIDI, TCP/IP, USB, Firewire) as well as inter-applications exchange protocols, he only needs to open up his program to allow others access to the parameters the way they like. He does not have to give away his sources or unveil the precious tricks that make his program sound good. All he should do is to provide an external access and sufficient information about it. This also counts for non-real-time programs where MIDI files, SDIF files (Sound Description Interchange Format) [WDK99, WS99, BBS00, SW00] or well structured and documented text files may allow exchange of information between programs running on various platforms under different operating systems. His program could then be successfully used for the DAFx fields that apply:

- Sound design for studio applications.
- Electro-acoustic composition.
- Expanding the possibility of an acoustical instrument (hyperinstrument, score following), thereby allowing the performer to acquire more control of digital audio effects applied to his instrument.
- Creating a new instrument: the key elements are expressivity, breathing life into electronically generated or transformed sounds. The addition of real-time gestural control often transforms a rather simple algorithm into a powerful performance instrument mainly because of the added expressivity.
- Interactive dance performances: the movements of the dancers might be used to trigger/transform/spatialize sounds and music.
- Interactive sound installations: defining how it will react to the visitors.

An easy and effective way to achieve this is by writing external modules or plug-ins for various environments, following the design rules standard to these applications.

And other developers will continue their work of devising better GUIs, proposing more elegantly formalized algorithmic environments, developing more effective feature extraction methods and providing improved gesture controllers. This cooperative approach prevents everyone from reinventing the wheel and ensures that users may choose their favorite working environment and gestural controllers, capitalizing on all the knowledge and experience they have gathered over the years, but still using the most up-to-date DAFx algorithms.

Sound and Music

- [m-Fin95] J. Fineberg: “Empreintes”. Premiered the 18th of May 1995 at IRCAM, Paris, France.
- [m-Mac] T. Machover: “Brain Opera”.
- [m-Mai97] M. Maiguashca: The Nagual. In Reading Castañeda. 1 CD + Booklet. Edition ZKM 3. Wergo 2053-2, 1997.
- [m-Noi00] M. Noiret: “In Between”, Interactive dance performance. Premiered the 11th of March 2000, Brussels, Belgium.
- [m-Ris] J.C. Risset: “Eight sketches for one pianist”. CD Neuma Electro Acoustic Music III, 450-87.
- [m-Sie97] W. Siegel, H. Saunders, and P. Fynne: Movement Study, Interactive dance, performed at ICMC97, Thessaloniki, Greece.
- [m-Xen95] I. Xenakis: “La légende d’Er”. CD Montaigne MO 78 2058, 1995.

Bibliography

- [AD00] D. Arfib and J. Dudon. A digital version of the photonic instrument. In *Proc. International Computer Music Conference*, pp. 288–290, Berlin, 2000.
- [All84] J.F. Allouis. Logiciels pour le système temps réel SYTER. Software for the real-time SYTER system. In *Proc. International Computer Music Conference*, pp. 27–28 and pp. 163–164, 1984.
- [All85] J.F. Allouis. Use of high speed microprocessors for digital synthesis. In J. Strawn and C. Roads (eds), *Foundations of Computer Music*, MIT Press, pp. 281–288, 1985.
- [AM81] J.F. Allouis and B. Maillard. Simulation par ordinateur du studio de composition électroacoustique et applications à la composition musicale. Festival International du Son Haute Fidélité (Conférences des journées d’études), pp. 23–42, 1981.

- [Arf99] D. Arfib. Visual representations for digital audio effects and their control. In *Proc. DAFX-99 Digital Audio Effects Workshop*, pp. 63–66, Trondheim, December 1999.
- [BBS00] M. de Boer, J. Bonada, and X. Serra. Using the sound description interchange format within the SMS applications. In *Proc. International Computer Music Conference*, pp. 190–193, Berlin, 2000.
- [Bey89] P. Beyls. The musical universe of cellular automata. In *Proc. International Computer Music Conference*, pp. 34–41, Columbus, 1989.
- [BM97] R. Boulanger and M. Mathews. The 1997 Mathews radio-baton and improvisation modes. In *Proc. International Computer Music Conference*, pp. 395–398, Thessaloniki, 1997.
- [Bon94] B. Bongers. The use of active tactile and force feedback in timbre controlling electronic instruments. In *Proc. International Computer Music Conference*, pp. 171–174, 1994.
- [Bou90] R. Boulanger. Conducting the MIDI Orchestra, Part 1: Interviews with Max Mathews, Barry Vercoe and Roger Dannenberg. *Computer Music Journal*, 14(2):34–46, 1990.
- [Cad88] C. Cadoz. Instrumental gesture and musical composition. In *Proc. International Computer Music Conference*, pp. 1–12, 1988.
- [Car94] B. Cariou. The aXi0 MIDI Controller. In *Proc. International Computer Music Conference*, pp. 163–166, Aarhus, 1994.
- [CCP00] A. Camurri, P. Coletta, M. Peri, M. Ricchetti, A. Ricci, R. Trocca, and G. Volpe. A real-time platform for interactive dance and music systems. In *Proc. International Computer Music Conference*, pp. 262–265, Berlin, 2000.
- [CCRV00] A. Camurri, P. Coletta, M. Ricchetti, and G. Volpe. Synthesis of expressive movement. In *Proc. International Computer Music Conference*, pp. 270–273, Berlin, 2000.
- [CGL98] C. Clozier, F. Giraudon, and J.C. Le Duc. Le Gmebaphone. *Proc. JIM98*, pp. E2:1–7, 1998.
- [Cha93] C. Chafe. Tactile audio feedback. In *Proc. International Computer Music Conference*, pp. 76–79, 1993.
- [Chu96] L. Chu. Haptic feedback in computer music performance. In *Proc. International Computer Music Conference*, pp. 57–58, 1996.
- [CLF90] C. Cadoz, L. Lizowski, and J.L. Florens. A modular feedback keyboard. *Computer Music Journal*, 14(2):47–51, 1990.

- [CLF93] C. Cadoz, A. Luciani, and J.-L. Florens. CORDIS-ANIMA: a modeling and simulation system for sound synthesis - the general formalism. *Computer Music Journal*, 17(1):19–29, 1993.
- [CRT99] A. Camurri, M. Ricchetti, and R. Trocca. EyesWeb - toward gesture and affect recognition in dance/music interactive systems. *Proc. IEEE Multimedia Systems '99*, Florence, 1999.
- [Dut91] P. Dutilleux. *Vers la machine à sculpter le son, modification en temps réel des caractéristiques fréquentielles et temporelles des sons*. PhD thesis, University of Aix-Marseille II, 1991.
- [FA00] A. Freed and R. Avizienis. A new music keyboard with continuous key-position sensing and high-speed communication. In *Proc. International Computer Music Conference*, pp. 515–516, Berlin, 2000.
- [Fuj99] T. Fujishima. A real-time chord recognition of musical sounds: a system using common lisp music. In *Proc. International Computer Music Conference*, pp. 464–467, Beijing, 1999.
- [Ges98] Y. Geslin. Sound and music transformation environments: a twenty year experiment at the “Groupe de Recherches Musicales”. In *Proc. DAFX-98 Digital Audio Effects Workshop*, pp. 241–248, Barcelona, November 1998.
- [GF98] N. Griffith and M. Fernström. Litefoot — a floor space for recording dance and controlling media. In *Proc. International Computer Music Conference*, pp. 475–481, Michigan, 1998.
- [Gil94] B. Gillespie. The virtual piano action: design and implementation. In *Proc. International Computer Music Conference*, pp. 167–170, Aarhus, 1994.
- [HAS92] L. Haken, R. Abdullah, and M. Smart. The continuum: a continuous music keyboard. In *Proc. International Computer Music Conference*, pp. 81–84, San Jose, 1992.
- [HP99] K.H. Hsiao and J. Paradiso. A new continuous multimodal musical controller using wireless magnetic tags. In *Proc. International Computer Music Conference*, pp. 24–27, Beijing, 1999.
- [IT99] T. Ilmonem and T. Takala. Conductor following with artificial neural networks. In *Proc. International Computer Music Conference*, pp. 367–370, Beijing, 1999.
- [KG89] D. Keane and P. Gross. The MIDI baton. In *Proc. International Computer Music Conference*, pp. 151–154, Columbus, 1989.
- [KL90] R.B. Knapp and H.S. Lusted. A bioelectric controller for computer music applications. *Computer Music Journal*, 14(1):42–47, 1990.

- [Kre90] V. Krefeld. The hand in the Web: an interview with Michel Waisvisz. *Computer Music Journal*, 14(2):28–33, 1990.
- [KW91] D. Keane and K. Wood. The MIDI baton III. In *Proc. International Computer Music Conference*, pp. 541–544, Montreal, 1991.
- [Lau98] S. de Laubier. The meta-instrument. *Computer Music Journal*, 22(1):25–29, 1998.
- [Loh86] H. Lohner. The UPIC system: a user’s report. *Computer Music Journal*, 10(4):42–49, 1986.
- [MC89] T. Machover and J. Chung. Hyperinstruments: musically intelligent and interactive performance and creativity systems. In *Proc. International Computer Music Conference*, pp. 186–190, Columbus, 1989.
- [Mei91] J.P. Meinadier. L’Interface utilisateur. Série informatique et stratégie, DUNOD, 1991.
- [MIDI] MIDI 1.0 Detailed specification. The International MIDI Association. <http://www.midi.org/>
- [MP97] T. Marrin and J. Paradiso. The digital baton: a versatile performance instrument. In *Proc. International Computer Music Conference*, pp. 313–316, Thessaloniki, 1997.
- [MR90] R.A. Moog and T.L. Rhea. Evolution of the keyboard interface: the Bösendorfer 290 SE recording piano and the Moog multiply-touch-sensitive keyboards. *Computer Music Journal*, 14(2):52–60, 1990.
- [Mul98] A.G.E. Mulder. *Design of Virtual Three-dimensional Instruments for Sound Control*. PhD thesis, Simon Fraser University, Burnaby, BC, Canada, 1998.
- [Nag98] Y. Nagashima. Biosensorfusion: new interfaces for interactive multimedia art. In *Proc. International Computer Music Conference*, pp. 129–132, Michigan, 1998.
- [Nic00] C. Nichols. The vBow: a haptic musical controller human-computer interface. In *Proc. International Computer Music Conference*, pp. 274–276, Berlin, 2000.
- [Nie99] S.H. Nielsen. Real-time control of audio effects. In *Proc. DAFX-99 Digital Audio Effects Workshop*, pp. 55–58, Trondheim, December 1999.
- [Par97] J. Paradiso. Electronic music interfaces: new ways to play. *IEEE Spectrum Magazine*, 34(12):18–30, December 1997. <http://web.media.mit.edu/~joep/>
- [PAZ98] M. Puckette, T. Apel, and D. Zicarelli. Real-time audio analysis tools for Pd and MSP. In *Proc. International Computer Music Conference*, pp. 109–112, Michigan, 1998.

- [PD98] F. Pachet and O. Delerue. Constraint-based spatialization. In *Proc. DAFX-98 Digital Audio Effects Workshop*, pp. 71–75, Barcelona, November 1998.
- [PG97] J. Paradiso and N. Gershenfeld. Musical applications of electric field sensing. *Computer Music Journal*, 21(3):69–89, 1997.
- [PHH99] J. Paradiso, K.H. Hsiao, and E. Hu. Interactive music for instrumented dancing shoes. In *Proc. International Computer Music Conference*, pp. 453–456, Beijing, 1999.
- [PHSR00] J. Paradiso, K.H. Hsiao, J. Strickon, and P. Rice. New sensor and music systems for large interactive surfaces. In *Proc. International Computer Music Conference*, pp. 277–280, Berlin, 2000.
- [PL92] M. Puckette and C. Lippe. Score following in practice. In *Proc. International Computer Music Conference*, pp. 182–185, San Francisco, 1992.
- [Puc91a] M. Puckette. FTS: a real-time monitor for multiprocessor music synthesis. *Computer Music Journal*, 15(3):58–67, 1991.
- [Puc91b] M. Puckette. Combining event and signal processing in the MAX graphical programming environment. *Computer Music Journal*, 15(3):68–74, 1991.
- [Puc95] M. Puckette. Score following using the sung voice. In *Proc. International Computer Music Conference*, pp. 175–178, Banff, 1995.
- [Ram95a] C. Ramstein. Les interfaces à retour de force. *Proc. International Symposium on Electronic Arts*, Montreal. pp. 236–239, 1995.
- [Ram95b] C. Ramstein. MUIS: multimodal user interface system with force-feedback and physical models. *Proc. INTERACT 95*, pp. 157-163, Lillehammer, Norway, 1995.
- [RD96] J.C. Risset and S.V. Duyne. Real-time performance interaction with a computer-controlled acoustic piano. *Computer Music Journal*, 20(1):62–75, 1996.
- [RH00] J. Rovan and V. Hayward. Typology of tactile sounds and their synthesis in gesture-driven computer music performance. In *Trends in Gestural Control of Music*. IRCAM, 2000
- [RM90] D. Rubine and P. McAvinney. Programmable finger tracking instrument controllers. *Computer Music Journal*, 14(1):26–41, 1990.
- [Ros90] D. Rosenboom. The performing brain. *Computer Music Journal*, 14(1):48–66, 1990.

- [RS78] L.R. Rabiner and R.W. Schafer. *Digital Processing of Speech Signals*. Prentice-Hall, 1978.
- [SDWR00] S. Serafin, R. Dudas, M.M. Wanderley, and X. Rodet. Gestural control of a real-time physical model of a bowed string instrument. In *Proc. International Computer Music Conference*, pp. 375–378, Berlin, 2000.
- [Sie99] W. Siegel. Two compositions for interactive dance. In *Proc. International Computer Music Conference*, pp. 56–59, Beijing, 1999. <http://www.daimi.aau.dk/~diem/dance.html>
- [SJ98] W. Siegel and J. Jacobsen. The challenge of interactive dance: an overview and case study. *Computer Music Journal*, 22(4):29–43, 1998.
- [Smi00] A. Smirnov. Music and gesture: sensor technologies in interactive music and the theremin-based space control system. In *Proc. International Computer Music Conference*, pp. 511–514, Berlin, 2000.
- [SW00] D. Schwarz and M. Wright. Extensions and applications of the SDIF sound description interchange format. In *Proc. International Computer Music Conference*, pp. 481–484, Berlin, 2000.
- [Tar97] L. Tarabella. Studio report of the computer music lab of cnuce/c.n.r. In *Proc. International Computer Music Conference*, pp. 86–88, Thessaloniki, 1997.
- [Ter91] D. Terrugi. *Manuel du stage SYTER*, INA-GRM, 1991.
- [Ter94] D. Terrugi. The Morpho concepts: trends in software for acousmatic music composition. In *Proc. International Computer Music Conference*, pp. 213–215, Aarhus, 1994.
- [Ter98] D. Teruggi. *Le système SYTER. Son histoire, ses développements, sa production musicale, ses implications dans le langage électroacoustique d'aujourd'hui*. PhD thesis, University of Paris 8, 1998.
- [TDF95] T. Todoroff, E. Daubresse, and J. Fineberg. Iana~: a real-time environment for analysis and extraction of frequency components of complex orchestral sounds and its application within a musical realization. In *Proc. International Computer Music Conference*, pp. 292–293, Banff, 1995.
- [TMS98] L. Tarabella, M. Magrini, and G. Scapellato. A system for recognizing shape, position and rotation of the hands. In *Proc. International Computer Music Conference*, pp. 288–291, Michigan, 1998.
- [Tod95] T. Todoroff. Instrument de synthèse granulaire dans Max/FTS. *Proc. International Symposium on Electronic Arts*, Montreal, pp. 292–296, 1995.

- [Tod00] T. Todoroff. Modules externes Max/MSP pour l'analyse, la transformation et la synthèse sonore et leurs applications pour la danse interactive. In *Proceedings of Colloque International Max/MSP, IMEB and ENSI*, Bourges, 2000.
- [Tro99] J. Tro. Aspects of control and perception. In *Proc. DAFX-99 Digital Audio Effects Workshop*, pp. 63–66, Trondheim, December 1999.
- [Tru90] B. Truax. Time-shifting of sampled sound with a real-time granulation technique. In *Proc. International Computer Music Conference*, pp. 104–107, Glasgow, 1990.
- [TSS82] E. Terhardt, G. Stoll, and M. Seewann. Algorithm for extraction of pitch and pitch salience from complex tonal signals. *J. Acoust. Soc. Am.*, 71(3):679–688, 1982.
- [TT98] T. Todoroff and C. Traube. Transformations sonores en temps réel dans MAX/FTS et interfaces graphiques pour le contrôle de ces processus. *Recherches et Applications en Informatique Musicale*, Editions Hermès, Collection Informatique Musicale, Chapter 12, 1998.
- [TTL97] T. Todoroff, C. Traube, and J.-M. Ledent. NeXTStep graphical interfaces to control sound processing and spatialization instruments. In *Proc. International Computer Music Conference*, pp. 325–328, Thessaloniki, 1997.
- [UV99] T. Ungvary and R. Vertegaal. The SensOrg: time-complexity and the design of a musical cyberinstrument. In *Proc. International Computer Music Conference*, pp. 363–366, Beijing, 1999.
- [Var96] B. Varga. *Conversations with Iannis Xenakis*. Faber and Faber Ltd., 1996.
- [VU95] R. Vertegaal and T. Ungvary. The Sentograph: input devices and the communication of bodily expression. In *Proc. International Computer Music Conference*, pp. 253–256, Banff, 1995.
- [VUK96] R. Vertegaal, T. Ungvary, and M. Kieslinger. Towards a musician's cockpit: transducer, feedback and musical function. In *Proc. International Computer Music Conference*, pp. 308–311, 1996.
- [Wan97] M.M. Wanderley. Les nouveaux gestes de la musique. IRCAM internal report, April 1997.
- [WB00] M. Wanderley and M. Battier (eds). *Trends in Gestural Control of Music*. IRCAM, 2000.
- [WDK99] M. Wright, R. Dudas, S. Khouri, R. Wang, and D. Zicarelli. Supporting the sound description interchange format in the Max/MSP environment. In *Proc. International Computer Music Conference*, pp. 182–185, Beijing, 1999.

- [Wes79] D. Wessel. Timbre space as a musical control structure. *Computer Music Journal*, 3(2):45–52, 1979.
- [Win98] T. Winkler. Motion-sensing music: artistic and technical challenges in two works for dance. In *Proc. International Computer Music Conference*, pp. 471–474, Michigan, 1998.
- [WS99] M. Wright and E.D. Scheirer. Cross-coding SDIF into MPEG-4 structured audio. In *Proc. International Computer Music Conference*, pp. 589–596, Beijing, 1999.
- [WVIR00] M.M. Wanderley, J.P. Viollet, F. Isart, and X. Rodet. On the choice of transducer technologies for specific musical functions. In *Proc. International Computer Music Conference*, pp. 244–247, Berlin, 2000.
- [WWF97] M. Wright, D. Wessel, and A. Freed. New musical control structures from standard gestural controllers. In *Proc. International Computer Music Conference*, pp. 387–390, Thessaloniki, 1997.
- [Xen92] I. Xenakis. *Formalized Music*. Pendragon Press, Harmonologia Serie, No. 6, 1992.
- [Zic97] D. Zicarelli. An extensible real-time signal processing environment for MAX. In *Proc. International Computer Music Conference*, pp. 463–466, Thessaloniki, 1997.

Chapter 13

Bitstream Signal Processing

M. Sandler, U. Zölzer

13.1 Introduction

The motivation for Bitstream Signal Processing (BSP) Sigma Delta Modulation (SDM) has become the predominant means of converting between analog and digital domains for audio. There are various reasons for this, principal among them initially at least, was the low cost of SDM Digital-to-analog Converters (DACs) providing a quality equivalent to Nyquist-rate DACs (based on resistor ladders or current sources). There are also good sonic arguments for the use of SDM data converters, currently mirrored in the debate of 96 kHz and 192 kHz sampling rates, which is that the benign and slow rate of roll-off of the anti-aliasing filters does not compromise the phase response. Either way, SDM converters at both the front end and the back end of audio systems are here to stay for the foreseeable future. So let us look at the conventional approach to audio signal processing where the converters are SDM, leaving aside for now the precise details of how SDM works. Referring to Fig. 13.1, we see that the analog signal coming in is converted to single-bit stream at a rate many times the Nyquist rate (typically 64 times, i.e. the sampling is at 64×48000 Hz). Because the processing is performed in what we shall call PCM format, i.e 16 or more bits at the Nyquist rate, there is a need to down-sample the bitstream. This is done using a special filter known as a Decimation filter — the combination of the blocks labelled O and LPF in Fig. 13.1.

Then the processing is performed as conventional DSP dictates, for example using programmable microprocessor with a Harvard or Super-Harvard architecture, such as those from Motorola, Texas Instruments or Analog Devices. It is performed at the Nyquist rate, f_s - the sampling frequency - on b bit signal samples. Then to convert it back to analog form, for monitoring or final rendition, we again use SDM, this time in a DAC. However, although the SDM DAC itself is very simple, it needs to run at the high, super-Nyquist rate (e.g. 64×48000 Hz again). This means that

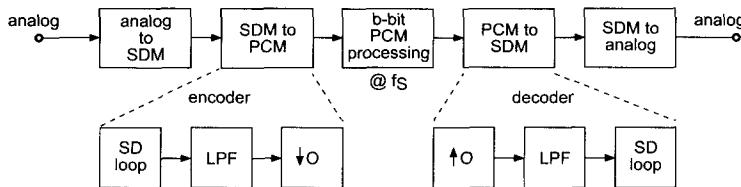


Figure 13.1 Basic structure of DSP system based on Sigma Delta interfaces to the analog world. SDM=Sigma Delta Modulation representation; LPF = lowpass filter; boxes with O are sample rate decrease and increase respectively, by a factor O, the over-sampling ratio.

the PCM format signal our processing has produced must have its sampling rate increased, using a special filter known as an interpolation filter - the combination of the blocks labelled “LPF” and “O” in the figure. So overall, we have simple analog-digital and digital-analog portals in our system, but we have the expense of the decimation and interpolation filters on top of this. If the PCM processing in between these is simple, there is likely to be a processing penalty, in the sense that the DSP MIPS are dominated by the interpolation and decimation filters.

Thus a question arises. Would it not be simpler (not to mention more elegant) if we could perform the processing directly on the SDM bitstream that comes out of the SDM ADC, and write it directly to the SDM DAC - cut out the middle man? This is shown in Fig. 13.2 where it is clear that we no longer use the sample rate decrease (decimator) and increase (interpolator) filters and thus have saved processing.

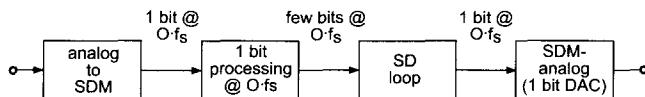


Figure 13.2 Basic structure of DSP system based on Sigma Delta coding and bitstream signal processing. The provision for the result of the operation being more than 1 bit wide is made explicit - the SD loop.

Probably it is reasoning of this sort that persuaded researchers at Sony and Philips to investigate how beneficial it would be to work entirely in the SDM signal domain. They have produced proposals and indeed products, based on the concept that signals should be stored, processed and generally manipulated in SDM bitstream format. This they have called Direct Stream Digital, and it is this representation that underpins the new Super Audio CD format. Because Sony and Philips propose not only selling music in the Super Audio format, but to configure the complete processing chain so that it is based on it, that shapes the remainder of this section. We will look first at the general principles of SDM in both ADCs and DACs, and will then look at two of the most common signal processing tasks, IIR and FIR filters, that can be accomplished in SDM format.

13.2 Sigma Delta Modulation

Conventional data converters, both ADCs (Analog-to-Digital Converters) and DACs (Digital-to-Analog Converters), use no (or mild) over-sampling and as such convert signals at or just over the Nyquist sampling limit. A general structure for a Nyquist ADC is shown in Fig. 13.3 and that of a Nyquist DAC is shown in Fig. 13.4. Typically, then, for a 20 kHz bandwidth, the sampling and conversion will take place at a rate of, say, 44.1 kHz for CD quality.

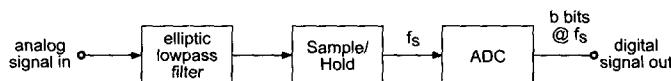


Figure 13.3 Nyquist-rate ADC.

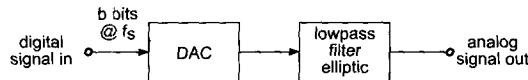


Figure 13.4 Nyquist-rate DAC.

A generic over-sampling ADC structure is shown in Fig. 13.5, where the key point to note is that the core ADC will typically produce fewer bits per sample than the final digital output. The extra bits in the output are regained by the decimation filter which reduces the sampling rate back down to (or close to) the Nyquist rate. The reason this works is that the quantization error power of a core ADC is determined solely by the number of bits it uses. However, with over-sampling, this power is spread more thinly through the spectrum, so that the noise power per Hz is reduced and in-band SNR is improved. Because the decimation filter is used to extract only the wanted baseband signal it is possible to recover the additional bits. The SNR improvement is 3 dB for every doubling of sampling rate (over and above Nyquist), or an extra bit for a 4 times over-sampling, 2 extra bits for 16 times over-sampling, etc. Over-sampling like this is a simple way to trade amplitude resolution for time resolution, but if non-linear techniques are used, improved gains can be made. Figure 13.6 presents a generic over-sampling DAC. The principal reason for choosing such a structure is to ease the specification of the reconstruction filter, which is analog and in Nyquist DACs may introduce unwanted phase distortion. The purpose of the interpolation filter is to increase the sampling rate of the incoming digital signal prior to conversion. The signal at the output of that block should approximate

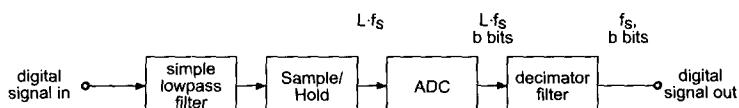


Figure 13.5 Over-sampling ADC.

closely the digital signal that would have been obtained had the analog signal been sampled at the higher rate rather than the Nyquist rate.

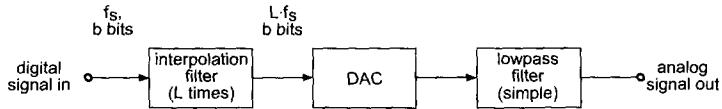


Figure 13.6 Over-sampling DAC.

Together, interpolation filters and decimation filters are known as multirate filters [SR73, CR83]. This is because they operate at more than one sampling frequency. It is by operating these non-linear SDM systems in excess of the Nyquist sampling rate that they are able to perform in a quasi-linear manner. Sigma Delta Modulation works by converting a signal into a sequence of pulses, whose short-term average follows that of the modulating signal [CT91, NST97]. In this case, the pulses are of fixed duration, and their number is proportional to the signal amplitude: it is thus sometimes referred to as Pulse Density Modulation (PDM) and has much in common with error diffusion techniques, as used in image half-toning. An example of the pulse stream that results from SDM is shown in Fig. 13.7, and a generic structure for a SDM is shown in Fig. 13.8. Note that the number of levels that a SDM pulse stream can attain is determined by the number of discrete levels of its internal quantizer. For a single bit quantizer, there are two levels, normally denoted as +1 and -1. Note that often it is clearer and/or more convenient to label the negative pulse value as 0, so that pulse values are 1 and 0.

SDM is closely related to delta modulation in which a single bit commands a decoder either to increase or decrease its output level - thus it is capable of tracking the input. However, SDM both simplifies the decoder and encoder by including an integrator in the encoder's feed-forward path. The development of SDM is generally attributed to Cutler [Cut52, Cut54], whose paper actually deals with an error diffusion approach. The structure now known as Sigma Delta Modulator is due to Inose *et al.* [IYM62, IY63] who first reported it in the early 1960s.

13.2.1 A Simple Linearized Model of SDM

The non-linearity within the SDM block diagram makes precise analysis somewhat intractable, though there have been some excellent publications on exact non-linear analysis of SDMs [FC91]. This approach leads to accurate but restricted results, so it is common to model the SDM by a linearized method in which the quantizer is replaced by an additive noise source. Figure 13.9 shows the linearized model.

Overall, the output, $O(z)$ is a combination of the quantization error, $q(n)$ and the feedback error, $I(z) - O(z)$, given by

$$O(z) = G(z) \cdot (I(z) - O(z)) + Q(z). \quad (13.1)$$

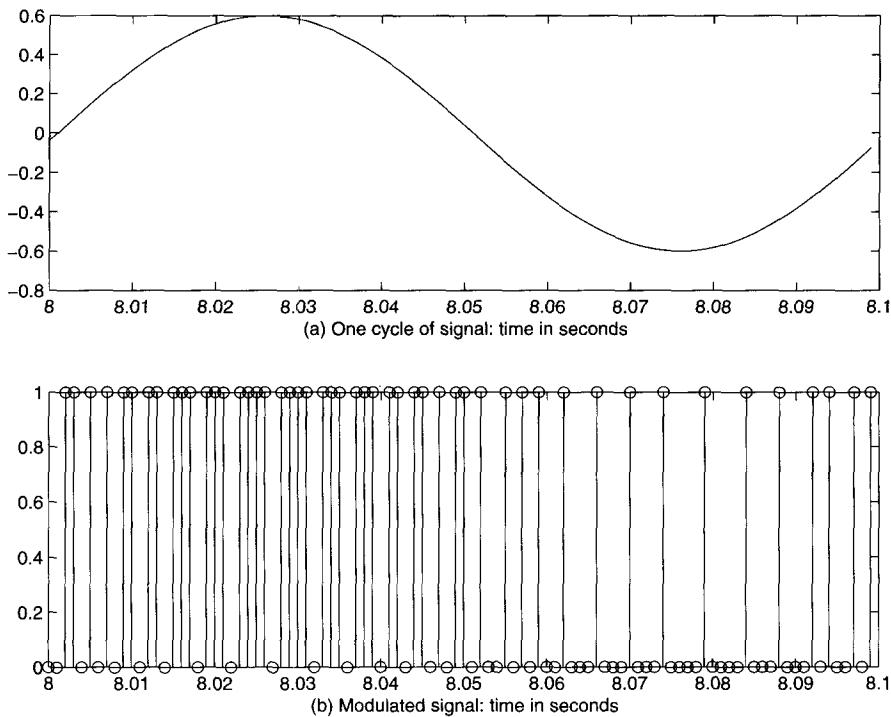


Figure 13.7 First-order Sigma Delta Modulation of 1 kHz sine wave, 50 times oversampled.

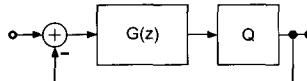


Figure 13.8 Sigma Delta Modulator.

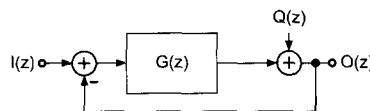


Figure 13.9 Linearized model of Sigma Delta Modulator suitable for analysis.

Setting $Q(z) = 0$ we find the output as a function of the input only, and leads to the Signal Transfer Function (STF)

$$\text{STF}(z) = \frac{O(z)}{I(z)} = \frac{G(z)}{1 + G(z)}. \quad (13.2)$$

Setting the input $I(z) = 0$ we find the Noise Transfer Function (NTF)

$$\text{NTF}(z) = \frac{O(z)}{Q(z)} = \frac{1}{1 + G(z)}. \quad (13.3)$$

Note that this makes no assumptions about the probability density function of $q(n)$ or indeed the nature of the quantizer or the amount of over-sampling. As the number of bits or levels in the quantizer increases so the analysis proves increasingly accurate. The same is true as the over-sampling increases.

In most cases, the design of a SDM will start from this simple form, which enables a first set of system parameters to be obtained. What then follows is an extensive process of simulating, testing and modifying until the system performs adequately well for the intended application. It is still a topic of research interest to be able to produce SDM models which are amenable to single pass design strategies for which no simulate-test cycles are needed.

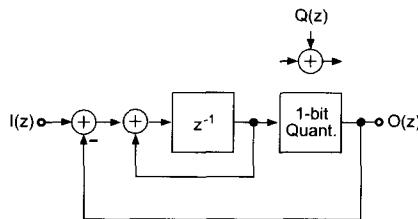


Figure 13.10 First-order Sigma Delta Modulator.

13.2.2 A First-order SDM System

In its simplest form, $G(z)$ is an integrator, which has high gain at low frequencies so that $\text{STF}(z) \approx 1$, whereas the NTF at low frequencies will be small, so that the quantization error (manifested as noise) is greatly suppressed. A first-order SDM is shown in Fig. 13.10. This is demonstrated in Fig. 13.11, which presents the spectra for the signal of Fig. 13.10. The noise suppression at baseband is clear and we can see that, in spite of only a single bit representation of each sample, the modulated representation provides a good signal-to-noise ratio over a useful bandwidth. A simple Matlab simulation of this system is shown in M-file 13.1.

```
M-file 13.1 (fo_sdm.m)
% First-order SDM
clear;clf;
N=10000; n=[0:N-1];
signal=0.6*sin(100*n*2*pi/N);
q=0;node3=0;

% main loop
for i=0:N-1,
```

```

q=(node3>0);
node1=signal(i+1)-2*(q-0.5);
node2=node1+node3;
node3=node2;
y(i+1)=q;
end

% plots
time1=8:0.001:8.999;time2=8:0.001:8.099;
freq1=0:1:499;freq2=0:0.01:49.99;
figure(1);
subplot(2,1,1);
plot(time2,signal(8000:8099));
xlabel('a) One cycle of signal : Time in seconds');
subplot(2,1,2);
stem(time2,y(8000:8099),'b');
xlabel('b) Modulated signal : Time in seconds');

[P,F]=spectrum(y,2048,1024,[],1000);
figure(2);
subplot(2,1,1);
plot(F,20*log10(P(:,1)+0.00001));
xlabel('a) Power Spectrum (Hz) up to half-sampling frequency');
subplot(2,1,2);
plot(F(1:205),20*log10(P(1:205,1)));
xlabel('b) Narrow-band Power Spectrum (Hz)');

```

13.2.3 Second and Higher Order SDM Systems

Higher order SDMs use cascaded integrators in the loop filter, which increase the noise suppression at low frequencies. The order of the filter is used to describe the order of the modulator, e.g. a third-order filter is used in a third-order modulator. A second-order SDM is shown in Fig. 13.12.

The effect of increasing order is demonstrated in Fig. 13.13 for a second-order modulator, with the same input signal and over-sampling rate (500) as Fig. 13.11. We can see greater baseband resolution and/or a wider usable bandwidth. For all SDMs this is at the cost of amplifying noise at high frequencies. A simple Matlab simulation of this system is shown in M-file 13.2.

```

M-file 13.2 (so_sdm.m)
% Second-order SDM
clear,clf;
N=10000;
n=[0:N-1];
signal=0.6*sin(10*n*2*pi/N);

```

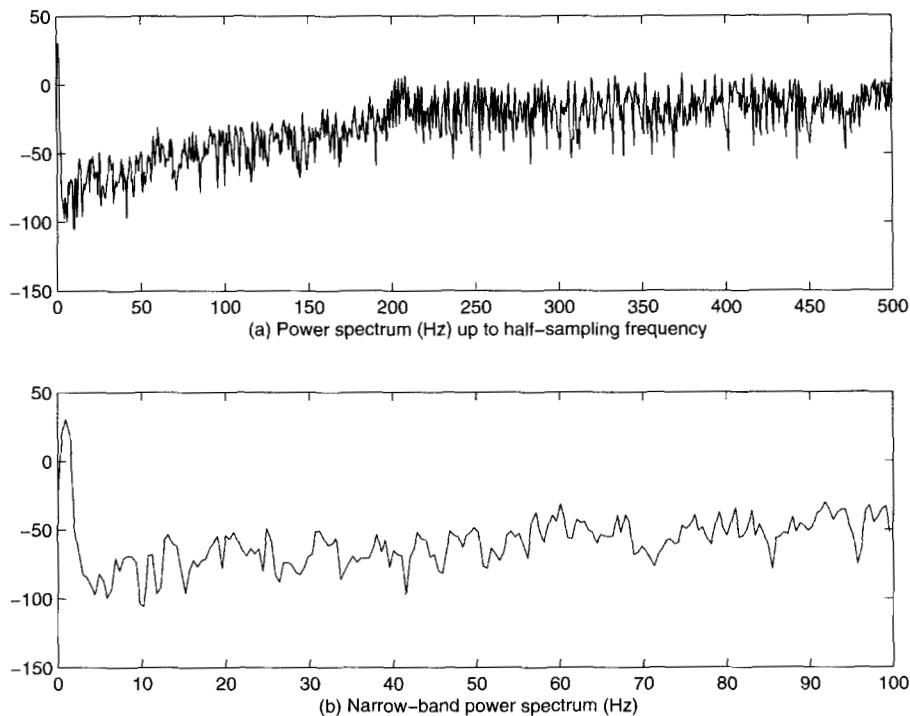


Figure 13.11 Power spectral analysis of first-order SDM for 500 times over-sampled signal. Note the good signal resolution at low frequencies in (a) magnified in (b).

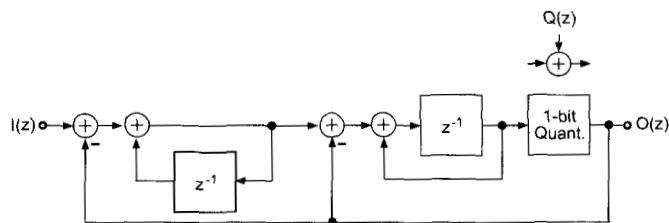


Figure 13.12 Second-order Sigma Delta Modulator.

```
q=0;node5=0;node2=0;node4=0;
```

```
% main loop
for i=0:N-1,
q=(node5>0);
node1=signal(i+1)-2*(q-0.5);
node2=node2+node1;
node3=node2-2*(q-0.5);
node4=node3+node5;
```

```
% q=(node5>0);
node5=node4;
y(i+1)=q;
end

% plots
figure(1);
subplot(4,1,1);
plot(n(N/2:N-1),signal(N/2:N-1));
subplot(4,1,2)
plot(n(N/2:N-1),y(N/2:N-1),'y');
subplot(4,1,3);
plot(n(N/2:N/2+199),signal(N/2:N/2+199));
subplot(4,1,4)
stem(n(N/2:N/2+199),y(N/2:N/2+199),'y');

[P,F]=spectrum(y,2048,1024,[],1000);
figure(2);
subplot(2,1,1);
plot(F,20*log10(P(:,1)));
xlabel('a) Power Spectrum (Hz) up to half-sampling frequency');
subplot(2,1,2);
plot(F(1:205),20*log10(P(1:205,1)));
xlabel('b) Narrow-band Power Spectrum (Hz)');
```

The generic structure of Fig. 13.7 is often known as an interpolative SDM. Many alternative topologies exist, but perhaps the most popular is the so-called multiple feedback structure. This is shown in Fig. 13.14. The great advantage of this structure is that it is tolerant to coefficient and data path quantization because the filter is constructed from a cascade of integrators.

13.3 BSP Filtering Concepts

Since the early 1990s a small but significant flurry of work has appeared, dealing with a variety of signal processing algorithms implemented to process a SDM bit-stream signal. Topics covered include: FIR filtering [WG90, Won92], second-order IIR filtering [JL93, JLC93, Qiu93], adaptive filtering [RKT92, QM93], high quality sinusoidal oscillators [LRJ93a, LRJ93b], signal mixing [Mal92], dynamic range compression and expansion [Qiu92], phase/frequency modulation [RC94] and de-modulation [BC94, Gal94]. Useful introductions to sigma-delta signal processing have appeared [MO91, Mal92, Dia94] and present a wide range of the potential applications.

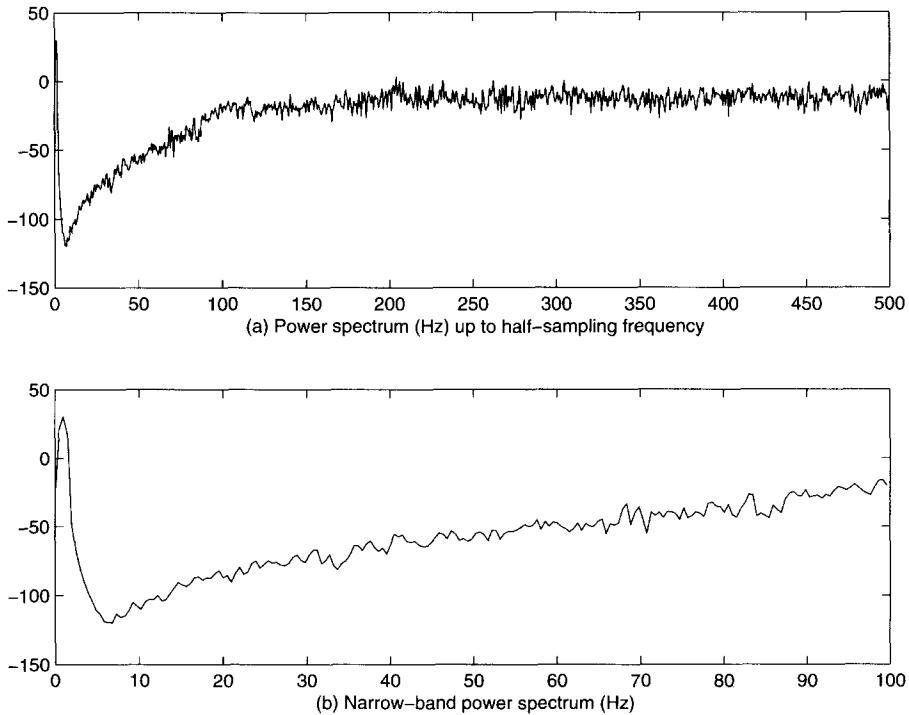


Figure 13.13 Power spectral analysis of second-order SDM for same input as Fig. 13.11. Note the improved signal resolution at baseband compared to the first-order modulator.

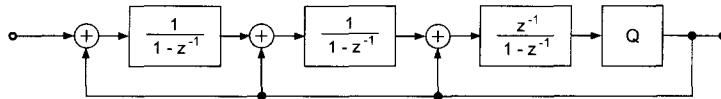


Figure 13.14 Alternative, multiloop SDM structure.

13.3.1 Addition and Multiplication of Bitstream Signals

In many audio applications it is necessary to add or subtract two bitstream signals or to multiply a bitstream signal by a constant parameter for gain manipulation. The output signal should be a bitstream signal. A bitstream adder was proposed in [OM90] and is shown in Fig. 13.15. Two over-sampled one-bit signals $x_1(n)$ and $x_2(n)$ are added, producing a two-bit result with the sum and the carry bit. The sum bit is stored and added to the following input bits and the carry bit forms the output bitstream. It can be shown [OM90], that for low frequencies the carry output signal represents the addition of the two input signals. The output signal

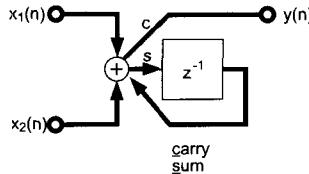


Figure 13.15 Summation of bitstream signals [OM90].

can be represented by its z-transform according to

$$Y(z) = X_1(z) + X_2(z) - \underbrace{\text{NTF}(z) \cdot \text{SUM}(z)}_{(1-z^{-1})}. \quad (13.4)$$

The equation represents a first-order sigma-delta modulator, where the sum signal corresponds to the quantization error. For low frequencies the term representing the error signal is reduced by the noise transfer function NTF. It is possible to perform subtraction by inverting the required input bitstream signal.

Several possible multiplication schemes for bitstream processing are discussed in [MO91, Mal92, Dia94]. A basic building block for filtering applications was proposed in [JL93] as a sigma-delta attenuator, which is shown in Fig. 13.16. A bitstream signal $x(n)$ at the over-sampled rate is multiplied by a multibit coefficient a_1 . The multiplier can be efficiently realized by a 2-input multiplexer which selects either a_1 or $-a_1$. The following SD modulator to return a one-bit output is a simplified second-order modulator [JL93] with low complexity.

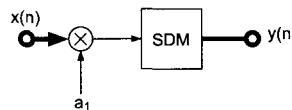


Figure 13.16 Multiplication of a bitstream signal by a sigma-delta attenuator [JL93].

13.3.2 SD IIR Filters

Equalizers for audio applications are mainly based on first- and second-order filters as discussed in Chapter 2. A first-order SD IIR filter was introduced in [JL93] and is shown in Fig. 13.17. It is based on the above described SD attenuator. The transfer function can be shown [Tsi00] to be given by

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_1 z^{-1}}{1 - (1 - a_1) z^{-1}}. \quad (13.5)$$

For realizing higher order filters a cascade of first- and second-order filters is used. A second-order SD IIR filter was also proposed in [JL93]. The block diagram

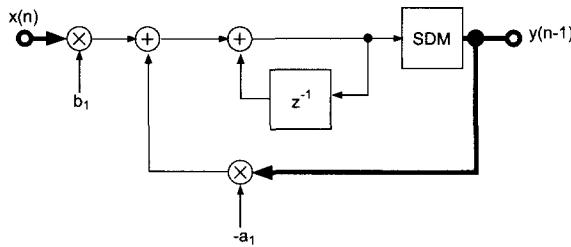


Figure 13.17 First-order SD IIR filter [JL93].

is shown in Fig. 13.18. The number of SD modulators is equal to the order of the IIR filter in order to keep the number noise sources as low as possible. The second-order SD IIR filter is based on an integrator based IIR filter. According to [JL93] the transfer function is given by

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_2 + (b_1 - 2b_2)z^{-1} + (b_0a_1 - b_1 + b_2)z^{-2}}{1 - (2 - a_2)z^{-1} + (1 + a_1^2 - a_2)z^{-2}}. \quad (13.6)$$

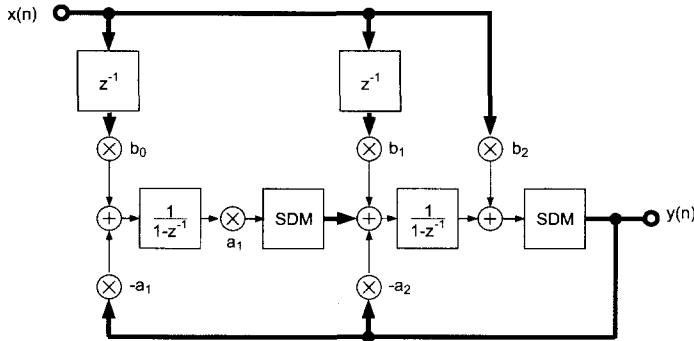


Figure 13.18 Second-order SD IIR filter [JL93].

The application of established IIR filter topologies for over-sampling SD IIR filters is straight forward. However, the inclusion of SD modulators to maintain a bitstream output signal has to be investigated carefully.

13.3.3 SD FIR Filters

FIR filters for bitstream processing have been discussed in several publications [PL73, WG90, Won92, KSSA96]. Two methods for building FIR digital filters are treated in [WG90, Won92] where either the impulse response or the input signal or both are encoded using sigma-delta modulation. Realization issues for a VLSI implementation of a sigma-delta bitstream FIR filter are discussed in [KSSA96]. The

resulting FIR filter topology is shown in Fig. 13.19. The input signal is a bitstream signal with one-bit representation and the coefficients are in multibit representation. The multipliers are again realized as 2-input multiplexers which select the coefficients b_i or $-b_i$. The number and values of the coefficients are identical to the prototype design for the Nyquist sampling rate f_s . The interpolation by oversampling factor R of the impulse response is achieved by replacing the z^{-1} operator by z^{-R} . This approach allows the design of the filter coefficients b_0, b_1, \dots, b_{N-1} at the sampling rate f_s using well-known design techniques.

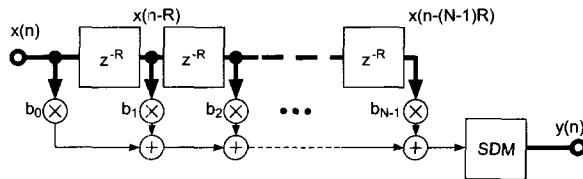


Figure 13.19 SD bitstream FIR filter [WG90, Won92, KSSA96].

13.4 Conclusion

We have reviewed the basics of SDM and shown how it generates a single bit representation of audio signals. Subsequently we provide an introduction to the ways in which direct bitstream processing of these signals can be performed.

Techniques like these are only really suitable for real-time hardware implementation where cost savings may be made by reducing chip count/size. For example, one might imagine a next generation of on-stage effects processors built around such devices. They have already been used in prototype mixing consoles. When implemented carefully the processing and the final sound can be every bit as good as by more conventional means. Whether or not these processing techniques become mainstream or not depends less on how good they are and more on the market success of the DSD format - only time will tell us the answer.

Bibliography

- [BC94] R.D. Beards and M.A. Copeland. An oversampling delta-sigma frequency discriminator. *IEEE Trans. on Circuits and Systems-II: Analog and Digital Signal Processing*, 41(1):26–32, January 1994.
- [CR83] R.E. Crochiere and L.R. Rabiner. *Multirate Digital Signal Processing*. Prentice-Hall, 1983.
- [CT91] J.C. Candy and G.C. Temes. *Oversampling Delta-Sigma Converters*. IEEE Press, 1991.

- [Cut52] C.C. Cutler. Differential quantization of communication signals. U.S. Patent 2 605 361, July 29, 1952.
- [Cut54] C. Cutler. Transmission systems employing quantization. U.S. Patent 2 927 962, March 8, 1960 (filed 1954).
- [Dia94] V. Dias. Sigma-delta signal processing. In *IEEE International Symposium on Circuits and Systems*, pp. 421–424, 1994.
- [FC91] O. Feely and L.O. Chua. The effect of integrator leak in sigma-delta modulation. *IEEE Trans. on Circuits and Systems*, 38(11):1293–1305, November 1991.
- [Gal94] I. Galton. High-order delta-sigma frequency-to-digital conversion. In *Proc. IEEE ISCAS*, pp. 441–444, 1994.
- [IY63] H. Inose and Y. Yasuda. A unity bit coding method by negative feedback. *Proc. IEEE*, 51:1524–1535, 1963.
- [IYM62] H. Inose, Y. Yasuda, and J. Murakami. A telemetering system by code modulation — delta-sigma modulation. *IRE Trans. Space Electron. Telemetry*, SET-8:204–209, September 1962.
- [JL93] D.A. Johns and D.M. Lewis. Design and analysis of delta-sigma based IIR filters. *IEEE Trans. on Circuits and Systems-II: Analog and Digital Signal Processing*, 40(4):233–240, April 1993.
- [JLC93] D.A. Johns, D.M. Lewis, and D. Cherapacha. Highly selective analog filters using delta-sigma based IIR filtering. In *Proc. IEEE ISCAS*, pp. 1302–1305, 1993.
- [KSSA96] S.M. Kershaw, S. Summerfield, M.B. Sandler, and M. Anderson. Realisation and implementation of a sigma-delta bitstream FIR filter. *IEE Proc. Circuits Devices Syst.*, 143(5):267–273, October 1996.
- [LRJ93a] A.K. Lu, G.W. Roberts, and D.A. Johns. A high quality analog oscillator using oversampling D/A conversion techniques. In *Proc. IEEE ISCAS*, pp. 1298–1301, 1993.
- [LRJ93b] A.K. Lu, G.W. Roberts, and D.A. Johns. A high quality analog oscillator using oversampling D/A conversion techniques. *IEEE Trans. on Circuits and Systems-II: Analog and Digital Signal Processing*, 41(7):437–444, 1993.
- [Mal92] F. Maloberti. Non conventional signal processing by the use of sigma-delta technique: a tutorial introduction. In *IEEE International Symposium on Circuits and Systems*, pp. 2645–2648, 1992.
- [MO91] F. Maloberti and P. O’Leary. Processing of signals in their oversampled delta-sigma domain. In *IEEE International Conference on Circuits and Systems*, pp. 438–441, 1991.

- [NST97] S. Norsworthy, R. Schreier, and G.C. Temes. *Delta Sigma Data Converters: Theory, Design and Simulation*. IEEE Press, New York, 1997.
- [OM90] P. O'Leary and F. Maloberti. Bitstream adder for oversampling coded data. *Electronics Letters*, 26(20):1708–1709, 27th September 1990.
- [PL73] A. Peled and B. Liu. A new approach to the realization of nonrecursive digital filters. *IEEE Trans. Audio Electroacoust.*, AU-21:477–484, December 1973.
- [Qiu92] H. Qiuting. Monolithic CMOS companders based on Sigma-delta oversampling. In *Proc. IEEE ISCAS*, pp. 2649–2652, 1992.
- [Qiu93] H. Qiuting. Linear phase filters configured as a combination of sigma-delta modulator, sc transversal filter and a low-q biquad. In *Proc. IEEE ISCAS*, pp. 1306–1309, 1993.
- [QM93] H. Quiting and G. Moschytz. Analog multiplierless LMS adaptive FIR filter structure. *IEEE Trans. on Circuits and Systems-II: Analog and Digital Signal Processing*, 40(12):790–793, Dec. 1993.
- [RC94] T.A.D. Riley and M.A. Copeland. A simplified continuous phase modulator. *IEEE Trans. on Circuits and Systems-II: Analog and Digital Signal Processing*, 41(5):321–328, May 1994.
- [RKT92] T. Ritonиеми, T. Karema, and H. Tenhunen. A sigma-delta modulation based analog-adaptive filter. In *Proc. IEEE ISCAS*, pp. 2657–2660, 1992.
- [SR73] R.W. Schafer and L.R. Rabiner. A digital signal processing approach to interpolation. *Proc. IEEE*, 61(6):692–702, June 1973.
- [Tsi00] E. Tsitsanis. An overview on sigma delta converters and sigma delta signal processing techniques. Master's thesis, King's College London, University of London, 2000.
- [WG90] P.W. Wong and R.M. Gray. FIR filters with sigma-delta modulation encoding. *IEEE Trans. on Acoustics, Speech and Signal Processing*, 38(6):979–990, June 1990.
- [Won92] P.W. Wong. Fully sigma-delta modulation encoded FIR filters. *IEEE Trans. on Acoustics, Speech and Signal Processing*, 40(6):1605–1610, June 1992.

Glossary

ADT	Automatic Double Tracking: A time-based signal processor that simulates the effect of playing a part, then overdubbing a second part to give a thicker sound.
Aliasing	Frequency components above half the sampling frequency of a sampled signal that are folded back into the audio spectrum (0-20 kHz).
AT constant	Time needed for a signal to reach 63 percent (-4dB) of its final amplitude. After three time constants it will have reached 95 percent (-0.4dB) of its final amplitude.
Attack time AT	Time for a signal to rise from 10 percent to 90 percent from its final amplitude.
Audio effect	A modification of a sound by use of a signal processing technique. It is sometimes called Audio-FX.
Auto pan	To change a signal's spatial position in the stereo field via some modulation source.
Brassage	French for time shuffling.
Chorus	Detuning effect where the original signal is mixed with a pitch modulated copy of the original signal. Pitch modulation is achieved by a random variation of the length of a delay line.
Click	A slight sharp noise, usually due to a discontinuity of the signal or to some computation error. In some forms of musical production, such as techno or live sampling, the clicks become such an important musical relevance, that they are even emphasized.
Clipping	Severe distortion of the signal because the amplitude is larger than the processing system can handle.
Comb filter	Filter effect occurring if the original signal is mixed with a delayed version of the original signal. The effect produces notches in the frequency domain at regular frequency intervals.

Compressor	A compressor is used for reducing the dynamics of an audio signal. Quiet parts or low levels of a signal are not modified but high levels or loud parts are reduced according to a static curve.
Controller	A device used to modify one or several parameters of an effect.
Convolution	Mathematical algorithm which is based on an input signal and another short signal (for example, an impulse response) and leads to an output signal.
Cross-synthesis	This effect takes two sound inputs and generates a third one which is a combination of the two input sounds. The general idea is to combine two sounds by spectrally shaping the first sound by the second one and preserving the pitch of the first sound.
Decay rate	The time rate at which a signal decreases in amplitude. Usually expressed in decibel per second (dB/s).
Decay time	Time for a signal to decrease from 90 percent to 10 percent from its initial amplitude.
De-emphasis	See pre-emphasis .
De-esser	A de-esser is a signal processing device for processing speech and vocals and is used to suppress high frequency sibilance.
Denoising	To decrease the noise within a sound.
Dispersion	Spreading a sound in time by a frequency-dependent time delay.
Distance rendering	The distance of a sound source is largely controllable by insertion of artificial wall reflections or reverberant room responses.
Distortion	A modification of the signal that is usually objectionable. When a signal is processed by a nonlinear system, some components appear that were not part of the original signal. They are called distortion products. Some musical instruments such as the electric guitar take advantage of distortions to enlarge and vary their timbre. This modifies the sound color by introducing nonlinear distortion products of the input signal. Related effects are Overdrive, Fuzz, Blender, Screamer.
Dithering	Adding a low-level noise to the signal before quantization. It improves the signal quality by decorrelating the quantification error and the signal.
Doppler effect	The Doppler effect raises the pitch of a sound source approaching the listener and lowers the pitch of a sound source departing the listener.

Dropout	A temporary loss of audio information. This is a typical problem of magnetic-tape-based storage and processing systems.
Dry	In general a “dry” sound is a sound that has not been processed by any means. It qualifies originally sounds that were recorded in an anechoic room. In our application the phrase “dry signal” denotes the sound before processing. See also wet .
Dubbing	Adding further material to an existing recording. Also known as overdubbing.
Ducking	A system for controlling the level of one audio signal with another. For example, background music can be made to “duck” whenever there is a voiceover [Whi99].
Echo	Several delayed versions of the original signal.
Equalizer	Filter system to shape the overall sound spectrum. Certain frequency ranges can be either increased or cut. A parametric equalizer allows individual setting of boost or cut, center frequency, bandwidth and filter type.
Exciter	Signal processor that emphasizes or de-emphasizes certain frequencies in order to change a signal’s timbre.
Expander	Expanders operate on low level signals and increase the dynamics of these low level signals.
Fade-in	Gradually increasing the amplitude of a signal from silence.
Fade-out	Gradually decreasing the amplitude of a signal to silence.
Feedback	To send some of an effect’s output signal back to the input. Also called regeneration.
Flanger	Sound effect occurring if the original signal is mixed with a delayed copy (less than 15 msec) of the original signal. The delay time is continuously varied with a low frequency sinusoid of 1 Hz.
Flatterzunge	A sound effect which is produced by rolling the tongue, blowing air through the mouth and performing a rapid fluttering motion of the tongue.
Flutter	Variations due to short-term speed variations at relatively rapid rates (above 6 Hz) [Met93]. See wow .
Foley	Imitation of real sounds for cinema applications. See also sound effect .
Formant changing	This effect produces a “Donald Duck” voice without any alteration of the fundamental frequency. It can be used for performing an alteration of a sound whenever there is a formant structure.

Freezing	(1) Selecting a fragment of sound and playing it as a loop. The time seems to be frozen to the date when the fragment was sampled. (2) Memorizing the spectrum envelope of a sound at a given time in order to apply this envelope onto another sound [Hal95, pp. 59-60].
Frequency shifter	A signal processor that translates all the frequency components of the signal by the same amount $f_i \rightarrow f_i + \Delta f$.
Frequency warping	A alteration of the linearity of the frequency axis.
FX	Shortcut for effects.
Gaussian noise	A random noise whose instantaneous amplitudes occur according to the Gaussian distribution.
Glissando	Linear transition from one pitch to another. This implies that the frequencies corresponding to the pitches vary according to a logarithmic law. See portamento .
Glitch	An unwanted short-term corruption of a signal, or the unexplained, short-term malfunction of a piece of equipment. See click .
Granulation	Extracting short segments from the input signal and rearranging them to synthesize complex new sounds.
Halaphon	A 4-channel sound projection system that was developed in 1971 by Hans Peter Haller and Peter Lawo. Four amplitude envelope oscillators with different waveforms driving four amplitude modulators allowed complex sound projection patterns at various speeds. An 8-channel version was used in 1973 for the production of “Explosante fixe” by Pierre Boulez and a 10-channel version for the production of “Prometeo” by Luigi Nono. The methods for spatialization proposed by John Chowning could also be implemented [Hal95, pp. 77-90].
Harmonizer	A trademark of Eventide for a pitch shifter.
Impulse response	The response of a system which is fed by an impulse signal.
Inharmonizer	This effect is obtained by frequency warping an original harmonic sound. The resulting sound is enriched by inharmonic partials.
Jitter	Degradation of a signal by sampling it at irregular sampling intervals. It can be interpreted as a modulation process where the audio signal equals the carrier and the jitter signal equals the modulation source.
Leslie	This effect was initially produced by rotating microphones or rotating loudspeakers. It can be approximated by a combination of tremolo and doppler effect.
Leveler	A dynamic processor that maintains (or “levels”) the amount of one audio signal based upon the level of a

second audio signal. Normally, the second signal is from an ambient noise sensing microphone. For example, a restaurant is a typical application where it is desired to maintain paging and background music a specified loudness above the ambient noise. The leveler monitors the background noise, dynamically increasing and decreasing the main audio signal as necessary to maintain a constant loudness differential between the two. Also called SPL controller [Boh00].

LFO	Low Frequency Oscillator. See modulation .
Limiter	Signal processor that lets the input signal pass through when its level is lower than a defined threshold and limits the output signal to a fixed level when the limiter threshold is exceeded.
Live sampling	A musical style that relies on the replay of sounds or fragments of them that are sampled during the performance from other performers or sound sources.
Masking	Phenomenon whereby one sound obscures another, usually one weaker and higher in frequency [Alt90].
Modulation	Process of altering a parameter, usually through some automatic or programmed means such as an LFO. See vibrato and tremolo .
Morphing	(1) Imposing a feature of one sound onto another. (2) A transition from one sound to another. (3) Generation of an intermediate sound between two others. (4) Generation of one sound out of the characteristics of another sounds. (5) Transforming one sound's spectrum into that of another. See spectral mutation .
Morphophone	A tape-based multi-delay system with a bandpass filter on the input signal as well as on each of the 10 playback heads. The mixed output can be fed back to the input. This device was designed by J. Poullin [Pou60] and A. Moles [Mol60, p. 73].
Multi-effects	A signal processor containing several different effects in a single package.
Mute	Cuts off a sound or reduce its level considerably.
Noise gate	Signal processor that lets the input signal pass through when its level is higher than a defined threshold.
Normalize	To amplify the sound so much that its maximum reaches the maximum level before clipping. This operation optimizes the use of the available dynamic range of the audio format and reduces the risk of corruption of the signal by low level perturbations that could happen during a further processing or the transmission of the sound.

Octavider	Producing a signal one octave below the input signal.
Off-line	A process is said to be off-line when it is applied on a recorded signal instead of on a real-time signal. Some processes are inherently off-line such as time contraction. Others are too computationally intensive to be performed in real-time.
Overdubbing	See dubbing .
Overload	To exceed the operating capacity of a representation, transmission or processing system.
Panorama	Composing a panorama of acoustic events in the space spanned by loudspeakers.
Patch	Another word for program, left over from the days of analog synthesizers. Also, the process of interconnecting various devices.
Peak filter	Tunable filter which boosts or cuts certain frequency bands with a bell-like frequency response.
Phasing	Effect where phase shifts of a copy of the original signal and mixing with the original signal cause phase cancellations and enhancements that sweep up and down the frequency axis.
Phonogène	A special tape recorder playing a loop at various speeds. It “has a circular arrangement of 12 capstan to change the tape speed within the 12 steps of the tempered scale”. The pinch roller facing each capstan is activated by a piano-like keyboard. This device was designed by P. Schaeffer. A further development of this device is called the “Phonogène universel”. It allows continuous transposition and/or time contraction and expansion. It relies on the rotating drum carrying 4 heads that was proposed by Springer [Mol60, p. 73]; [Sch73, p. 47]; [Pou60, Bod84].
Pink noise	Noise which has a continuous frequency spectrum and where each frequency band of constant relative bandwidth $\Delta f/f$ contains the same power. e.g. each octave has the same power.
Pitch	Subjective perception of frequency.
Pitch scaling	See pitch shifting .
Pitch shifting	Modification of the pitch of a signal. All the frequency components of the signal are multiplied by the same ratio. $f_i \rightarrow r \cdot f_i$. Asynchronous pitch shifting is achieved by varying the output sampling rate [Mas98] (see section 7.2).
Pitch transposer	A signal processor that duplicates the input at a defined pitch interval.

Portamento	A gliding effect where the pitch of a sound is changed gradually rather than abruptly when a pitch modification is required.
Post-echo	See print through .
Precedence effect	In a stereo loudspeaker set-up, if we step to one side of the central position and listen to a monophonic music program, we locate the apparent sound source in the same position as our closest loudspeaker, and the apparent position does not move even if the other channel is significantly louder.
Pre-echo	See print through .
Pre-emphasis	A system to boost high frequencies of a sound before processing it. A de-emphasis should be performed before playing the sound back after processing. This procedure attenuates high frequency noise contributed by the processing or transmission system.
Print through	The undesirable process that causes some magnetic information from a recorded analogue tape to become imprinted onto an adjacent layer. This can produce low-level pre- or post-echoes.
Quantize	Coding the amplitude of a signal with a given number of bits. Reducing the number of bits used to represent a signal usually degrades the quality of the signal. This effect can be attenuated by the use of dithering. Quantizing and dithering occur usually at the AD and DA stages of an audio processing system.
Random noise	A noise whose amplitude cannot be predicted precisely at any given time.
Ratio	Quotient of two quantities having the same unit. The transposition ratio is the quotient of the output frequencies to the input frequencies when they are expressed in Hz. The compression or expansion ratio is the quotient of the output amplitudes to the input amplitude when they are expressed in dB.
Real-time	A process is said to be real-time when it processes sound in the moment when it appears. A real-time system is fast enough to perform all the necessary computations to process one sample of sound within a sampling period.
Recirculate	See feedback .
Regeneration	See feedback .
Release time RT	Time for a signal to decrease from 90 percent to 10 percent from its final amplitude.
Resonator	Narrow bandwidth filter that amplifies frequencies around a center frequency.

Reverberation	Natural phenomenon occurring when sound waves propagate in an enclosed space.
Rise time	Time for a signal to rise from 10 percent to 90 percent from its final amplitude.
Robotization	Applying a fixed pitch onto a sound.
RT constant	Time needed for a signal to reach 37 percent (-9dB) of its initial amplitude. After 5 time constants it will have reached 1 percent (-43dB) of its initial amplitude.
Sampler	A digital system for recording and playing back short musical sounds in real-time. It is controlled by a MIDI keyboard or controller.
Scaling	As applied to continuous controllers, this determines how far a parameter will vary from the programmed setting in response to a given amount of controller change.
Shelving filter	Tunable filter which boosts or cuts the lower/higher end of the audio spectrum.
Shuffling	Out of a sequence of time or frequency elements of sound, producing a new sound with a new random order. The time shuffling is called <i>brassage</i> in french.
Sibilance	High frequency whistling or lisping sound that affects vocal recordings, due either to poor mic technique, excessive equalization or exaggerated vocal characteristics [Whi99].
Side-chain	In a signal processing circuit, such as one employing a VCA, a secondary signal path in parallel with the main signal path in which the condition or parameter of an audio signal that will cause a processor to begin working is sensed or detected. Typical applications use the side-chain information to control the gain of a VCA. The circuit may detect level or frequency or both. Devices utilizing side-chains for control generally fall into the classification of dynamic controllers [Boh00].
Side-chain input	The side chain input is necessary for the “ducking” effect, used by disc jockeys to automatically compress the music when they are talking [Whi99].
Side-chain insert	This insert can be used to insert an additional equalizer into the side chain, to turn a standard compressor into a de-esser for example [Whi99].
Slapback	Echo effect where only one replica of the original signal is produced.
Sound effect	A sound that comes as an audible illustration in an audio-visual or multi-media production.
Speaker emulator	A signal processor designed to imitate the effect of running a signal through a guitar amplifier cabinet.

Spectral mutation	Timbral interpolation between two sounds, the source sound and the target sound, in order to produce a third sound, the mutant. Operates on the phase and magnitude data pair of each frequency band of the source and target spectra [PE96].
Spectrum inverter	An amplitude modulator where the modulating frequency is equal to $f_s/2$. By usual audio sampling frequencies, this effect is usually unpleasant because most of the energy of the signal is located close to the higher limit of the frequency range.
Sweetening	Enhancing the sound of a recording with equalization and various other signal-processing techniques, usually during editing and mixing of a production.
Time-constant	A time required by a quantity that varies exponentially with time, but less any constant component, to change by the factor $1/e = 0.3679$. The quantity has reached 99 percent of its final value after a 5 time-constants.
Time warping	An alteration of the linearity of the time axis.
Transposition	See pitch shifting .
Tremolo	A slow periodic amplitude variation, at a typical rate of 0.5 to 20 Hz.
Undersampling	Sampling a signal at a frequency lower than twice the signal bandwidth. It produces aliasing.
Varispeed	Playing back a signal with time-varying speed.
VCA	Voltage Controlled Amplifier.
Vibrato	A cyclical pitch variation at a frequency of a few herz, typically 3 to 8 Hz.
Vocal gender change	Changing the gender of a given vocal sound.
Vocoding	See cross-synthesis .
Wah-wah	A foot-controlled signal processor containing a bandpass filter with variable center frequency. Moving the pedal back and forth changes the center frequency of the bandpass.
Wet	In practice the sound processed by an audio effect is often mixed to the initial sound. In this case, the processed sound is called the “wet signal” whereas the initial signal is called the “dry signal”. The term “wet” was initially used to qualify sounds affected by a lot of reverberation, whether contributed by a room or by an audio processor.
Whisperization	Applying a whisper effect onto a sound.
White noise	A sound whose power spectral density is essentially independent of frequency (white noise need not be random noise).

- Wow** Instantaneous variation of speed at moderately slow rates.
See **flutter**.
- Zigzag** During a zigzag process, a sound is played at the nominal speed but alternatively forwards and backwards. The reversal points are set by the performer [Wis94, Mir98].
- Zipper noise** Audible steps that occur when a parameter is being varied in a digital audio effect [Whi99].

Bibliography

- [Alt90] S.R. Alten. *Audio in Media*. Wadsworth, 1990.
- [Bod84] H. Bode. History of electronic sound modification. *J. Audio Eng. Soc.*, 32(10):730–739, October 1984.
- [Boh00] D.A. Bohn. <http://www.rane.com/digi-dic.htm>. *Rane Professional Audio Reference*, 2000.
- [Hal95] H.P. Haller. *Das Experimental Studio der Heinrich-Strobel-Stiftung des Südwestfunks Freiburg 1971–1989, Die Erforschung der Elektronischen Klangumformung und ihre Geschichte*. Nomos, 1995.
- [Mas98] D.C. Massie. Wavetable sampling synthesis. In M. Kahrs and K.-H. Brandenburg (eds), *Applications of Digital Signal Processing to Audio and Acoustics*, pp. 311–341. Kluwer, 1998.
- [Met93] B. Metzler. *Audio Measurement Handbook*. Audio Precision Inc., 1993.
- [Mir98] E.R. Miranda. *Computer Sound Synthesis for the Electronic Musician*. Focal Press, 1998.
- [Mol60] A. Moles. *Les musiques expérimentales*. Trad. D. Charles. Cercle d’Art Contemporain, 1960.
- [PE96] L. Polansky and T. Erbe. Spectral mutation in soundhack. *Computer Music Journal*, 20(1):92–101, Spring 1996.
- [Pou60] J. Poullin. Les chaines électro-acoustiques. *Flammarion*, pp. 229–239, September–December 1960.
- [Sch73] P. Schaeffer. *La musique concrète*. QSJ No 1287, PUF 1973.
- [Whi99] P. White. *Creative Recording, Effects and Processors*. Sanctuary Publishing, 1999.
- [Wis94] T. Wishart. *Audible Design: A Plain and Easy Introduction to Practical Sound Composition*. Orpheus the Pantomime, York, 1994.

Index

- Acoustic rays, 169, 172
- ADC, 3, 6, 501
 - Nyquist, 501
 - over-sampling, 501
 - sigma-delta, 500
- Additive synthesis, 379, 390, 396, 403, 404
- Algorithm, 6, 8, 10, 18–20, 24, 27, 29
- Aliasing distortion, 105, 108, 109
- AM-detector, 85
- Ambisonics, 141, 159, 163–164, 167
- Amplitude, 229
 - instantaneous, 404
 - time-varying, 242
- Amplitude envelope, 361–362, 366
- Amplitude follower, 82, 85, 88–90
- Amplitude modulation, 75, 77, 87, 90, 201, 220
- Amplitude panning, 139, 140, 162
- Amplitude scaler, 84
- Amplitude tracking, 477
- Analog-to-digital converter, *see* ADC
- Analysis, 237, 238, 242–244, 269, 277, 282, 294
 - grid, 269
 - hop size, 243, 244, 255, 269, 270, 277, 282
 - window, 239, 244
 - zero-phase, 244
- Apparent distance, 143, 187
- Apparent position, 138, 141, 170, 188, 190
- Apparent source width, 137, 153, 177, 187
- Architecture and music, 145
- Artificial reverberation, 152, 177, 180, 183
- Attack time, 95, 98, 99, 101, 102
- Attack time-constant, 84
- Autocorrelation, 307, 350–357, 366, 367
- Autocorrelation analysis, 351
- Autocorrelation features, 366–368
- Autocorrelation method, 305–308, 310, 350
- Averagers, 83
- Bandpass signal, 240–242, 247
- Bandwidth, 3, 317
- Baseband signal, 240–242, 247
- Bidirectional Reflection Distribution Function, *see* BRDF
- Binaural, 151, 153, 158–160, 165, 166, 187–190
- Binaural listening, 153, 158, 165
- Binaural model, 151, 189
- Binaural to transaural conversion, 166
- Bitstream, 499, 500
 - addition, 508
 - FIR filter, 507, 510
 - IIR filter, 509
 - multiplication, 508
 - signal processing, 500, 507
- Blumlein law, 138
- BRDF, 174
- Brilliance, 149, 175
- Buffer centering, 382
- Causality, 21
- Cepstrum, 300, 301, 310–315, 319–322, 326, 334, 347
 - complex, 311, 315

- real, 311, 315
- Cepstrum analysis, 311, 319, 323, 326
- Channel vocoder, 300–303, 315–317, 322
- Characteristic curve, 95, 111, 112, 117–120, 122, 123, 125, 128
- Chorus, 69, 70, 75, 460
- Circulant matrix, 183
- Clipping, 105, 112, 121, 122
 - asymmetrical, 112, 120–125
 - center, 353
 - hard, 105, 120
 - soft, 112, 114, 118–120, 123–125
 - symmetrical, 112, 118–120, 125
- Comb filter, 166, 172, 177, 178, 459
 - FIR, 63, 144
 - IIR, 64, 180
 - lowpass IIR, 72, 166, 178, 179
 - universal, 65
- Compression, 97, 100, 110, 116, 118, 120, 128
- Compression factor, 97
- Compressor, 97, 98, 100–102, 104, 105, 129
- Cone of confusion, 150
- Control, 465
 - algorithmic, 476
 - feedback loop, 467
 - force-feedback, 484
 - gestural interfaces, 478
 - GUI, 470
 - mapping, 467
 - MIDI, 479
 - sound features, 476
- Controllers
 - batons, 482
 - flat tracking devices, 483
 - force-feedback, 484
 - haptic interfaces, 484
 - hyperinstrument, 482
 - keyboards, 480
 - percussion interfaces, 481
 - string instruments, 481
 - wind instruments, 481
 - without physical contact, 486
 - worn on body, 485
- Convolution, 18, 19, 29, 48–50, 154, 175, 184–186, 192, 240, 255, 258, 264, 265, 408
- circular, 264, 274, 302, 303, 321
- fast, 6, 46, 49, 264, 265, 319, 323, 334
- Cross-correlation, 187, 209, 210
- Cross-synthesis, 285, 315–322, 478
- Csound, 158, 183
- DAC, 3, 6, 501
 - Nyquist, 501
 - over-sampling, 502
 - sigma-delta, 500
- DAFX, 1, 3, 29
- Decorrelation, 152, 188, 189
- Delay, 63–73, 143, 147, 169, 172, 173, 177, 178, 180–184, 352
 - dispersive, 445
 - fractional, 66–68
 - time-varying, 220
 - variable-length, 66, 70, 71, 142
- Delay line modulation, 82, 87, 201, 203, 220, 221
- Delay matrix, 182
- Demodulation, 82, 88
- Demodulator, 75, 82–85
- Denoising, 291–294
- Detector, 82, 83
 - amplitude, 82, 361
 - full-wave rectifier, 83, 85
 - half-wave rectifier, 83, 85
 - instantaneous envelope, 83–85
 - pitch, 76
 - RMS, 84, 85, 89
 - squarer, 83
- Deterministic component (see Sinusoidal component), 377, 396
- DFT, 7, 379
- Difference equation, 22, 23, 26, 29
- Diffusion, 171, 174, 184, 188, 189
- Digital signals, 3
- Digital systems, 2, 3, 18–23
- Digital Waveguide Networks, 178
- Digital-to-analog converter, *see DAC*

- Directional psychoacoustics, 141, 161–164
Directivity, 137, 192
Discrete Fourier transform, *see* DFT
discrete-time, 3, 6, 10, 21
Dispersion, 266–268
Distance rendering, 143–145
Distortion, 93, 113, 115–117, 120, 124–126, 128, 129, 131
Doppler effect, 68, 86, 87, 145–147, 169
Dummy head, 154, 186
Duration, 201, 202, 204, 205, 207, 216, 217, 227, 229, 232, 233
Dynamic behavior, 98
Dynamic range controller, 95–100
Dynamics, 99, 102
Dynamics processing, 95–105
- Early reflections, 175, 176, 178–180
Echo, 69
Enhancer, 131–132
Envelope detector, 95, 96, 98
Envelope follower, 95
Envelopment, 175–177, 185
Equalizer, 50–54
 time-varying, 58–59
Excitation signal, 305, 317, 328, 336
Exciter, 128–131
Expander, 97, 98, 100–102, 104
Expansion, 97, 100
Externalization, 151, 153, 154, 158, 187
- Far field, 171
Fast Fourier transform, *see* FFT
FDN, 178, 180–184, 189
Feature analysis, 399
Feature extraction, 336–369, 477
Feedback canceller, 59
Feedback Delay Network, *see* FDN
Feedback matrix, 182
FFT, 6–8, 10, 12, 15, 16, 238, 240, 242–244, 246, 251, 252, 254, 255, 257, 258, 262, 264, 265, 268, 269, 272–274, 279, 287, 294, 303, 310, 311, 313, 315, 323, 326, 330, 334, 337, 361, 363, 367, 382, 430, 433
FFT analysis, 244, 255, 263, 269
FFT filtering, 264, 265, 302
Filter, 31
 allpass, 32, 38–42, 51, 53, 57, 58, 155, 177–179, 188, 433
 arbitrary resolution, 416
 bandpass, 32, 35, 37, 38, 41, 43, 44, 47, 55, 56, 58, 240, 244, 247, 301, 317, 416
 bandreject, 32, 38, 41, 43, 44, 47
 bandwidth, 32, 38, 41, 42, 45, 50, 54, 55, 58
 comb, 172, 177, 178, 180
 complementary, 71
 complex-valued bandpass, 249
 damping factor, 33–38
 FIR comb, 63–65, 144
 gain, 50, 52, 54, 55, 58
 heterodyne, 246, 248, 255
 highpass, 32, 35, 37, 38, 40, 41, 43, 47
 IIR comb, 64
 lowpass, 3, 31, 33–38, 40, 43, 47, 240, 242
 notch, 32, 56, 57, 59
 peak, 50–55, 58
 Q factor, 50, 54, 55, 58
 resonator, 32
 shelving, 50–53, 58
 time-varying, 55–57, 397, 408
 universal comb, 65–66
 wah-wah, 55
Filter bank, 72, 240–242, 244–249, 254, 255, 269, 277, 301, 315
Filter bank summation model, 240, 244
FIR filter, 45–48, 304, 305, 352, 361
FIR system, 26
Flanger, 69, 75, 88
Flanging, 460
Flatter echoes, 63
Flatterzunge, 90, 460
Formant changing, 321–328
Formant move, 330, 331, 333, 334
Formant preservation, 215, 222

- Formant scaling, 227
- Formants, 204, 215, 222, 224, 225
- Frequency, 229, 248
 - bin, 240, 244, 255, 261–263, 268, 272, 277, 282, 284, 365
 - carrier, 76, 77, 80
 - center, 54, 55, 58, 247, 301
 - cut-off, 31, 33–35, 37–39, 41–43, 45, 50, 52, 54, 55
 - instantaneous, 247, 261, 263, 269, 272, 274, 277, 282, 284, 404
- Frequency band, 241, 242, 263, 266, 300
- Frequency modulation, 80
- Frequency resolution, 10, 337, 338, 342, 349, 380, 382, 383, 396, 406, 433
- Frequency response, 20, 33, 37, 38, 47, 53–55, 313, 322
- Frequency scaling, 322, 417
- Frequency shifting, 276
- Frequency warping, 154, 441
 - time-varying, 453
- Fundamental frequency, 220, 289, 308, 310, 312, 321, 337–339, 347–351, 377, 387, 393, 400, 402
- Fuzz, 116, 117, 120–122, 125, 127
- Gabor transform, 257
- Gaboret, 251, 257–259, 268, 282, 294
- Gaboret analysis and synthesis, 259
- Gaboret approach, 257, 258
- Gain factor, 95, 96, 99, 100, 102, 104, 128
- Gender change, 422
- Geometrical acoustics, 172
- Gestural interfaces, 478
- Glissando, 228, 460
- Granulation, 229–232
- Group delay, 39–41, 43
- Guide, 392, 393
- Halaphon, 170
- Halftone factor, 349
- Harmonic, 299, 336, 338, 339, 347, 350, 357, 362, 363, 366, 367, 377, 387, 392, 393, 400
- Harmonic distortion, 93–95, 105
- Harmonic generation, 126
- Harmonic/non-harmonic content, 366
- Harmonics, 94, 95, 105, 110, 112, 117, 119–122, 124, 126–128, 131
 - even order, 120–122, 124, 126
 - odd order, 112, 119, 121, 122, 124
- Harmonizer, 205, 215–217, 423
- Head model, 151
- Head shadowing, 156
- Head-Related Impulse Responses, *see* HRIR
- Head-Related Transfer Functions, *see* HRTF
- Heaviness, 176
- Hidden Markov model, 393, 426, 428
- Hilbert filter, 78, 79, 86
- Hilbert transform, 78, 79, 83
- Hoarseness, 424
- Holophonic reconstruction, 159
- Holophony, 163, 164
- Home theater, 164
- Homomorphic signal processing, 319
- Hop size, 242–245, 252, 255, 269, 270, 272, 273, 275, 277, 279, 282, 287, 289, 322, 338, 339, 379, 382, 401
- HRIR, 150, 154, 155
- HRTF, 150, 151, 153–159, 186
- IDFT, 10
- IFFT, 10, 238, 243, 244, 246, 251, 252, 254, 257, 264, 265, 268, 269, 272, 274, 279, 282, 291, 311, 313, 323, 330, 334, 361, 363, 367, 403–405, 430
- IFFT synthesis, 263, 287
- IID, 150, 151, 153, 176
- IIR filter, 38, 48, 305, 315–317
- IIR system, 22
- Image method, 173, 174
- Impression of distance, 143, 176
- Impulse response, 18, 20–22, 26, 27, 29, 46–49, 314
- Infinite limiter, 105

- Inharmonizer, 458
Interaural differences, 150–151, 189
Interaural intensity differences, *see* IID
Interaural time differences, *see* ITD
Interpolation, 67, 242, 255, 261, 269, 270, 280, 282, 289, 322, 326, 331, 333, 334, 382–384, 397, 398, 404, 424, 428
Inverse Discrete Fourier transform, *see* IDFT
Inverse Fast Fourier transform, *see* IFFT
Inverse filter, 305, 308
Irradiance, 174
ITD, 150, 151, 153, 155, 156, 176
- Laguerre transform, 448
short-time, 449
Lambertian diffusor, 174
Late reverberance, 176
Leakage effect, 11
Leslie effect, *see* Rotary loudspeaker
Level measurement, 95, 98, 99
Limiter, 97–101, 104, 105
Limiting, 97, 99, 105, 122, 129
Linear prediction, 300, 317–319, 356
Linear Predictive Coding, *see* LPC
Liveness, 176
Localization, 138, 141, 149–151, 153, 154, 157, 161, 169
Localization blur, 154
Localization with multiple loudspeakers, 160–161
Long-term Prediction (LTP), 351–360
Lossless prototype, 182, 183
LPC, 303–310, 315, 317, 318, 322, 336, 350, 399, 424
LPC analysis, 350
LPC analysis/synthesis structure, 304
LPC filter, 305, 306, 310
LTI system, 18
- Magnitude, 237, 240, 244, 251, 255, 269, 272, 285, 287, 291
random, 291
Magnitude processing, 247
- Magnitude response, 27, 39–41, 43, 52, 93, 301, 313
Magnitude spectrum, 7, 9, 10, 240, 291
Main lobe, 380, 381, 405, 406, 408
bandwidth, 381
MATLAB, 1
Maximum-Length Sequence, *see* MLS
MIDI controllers, 479
Mimicking, 365, 366
Mirror reflection, 173, 175
MLS, 154
Modulation, 75–82, 86–88, 361, 478
amplitude, 75, 77, 87, 90
frequency, 80
phase, 75, 80, 82, 88
Modulator, 76–82
amplitude, 77
frequency and phase, 80
ring, 76
single-side band, 77
Moorer’s reverberator, 179
Morphing, 88, 285, 424, 426, 460
Multiband effects, 71, 72
Mutation, 285–287
- Near field, 171
Noise gate, 97, 98, 102–104, 291, 292
Nonlinear distortion, *see* Distortion
Nonlinear modeling, 106–109
Nonlinear processing, 93–135
Nonlinear system, 93, 94, 106–108
Nonlinearity, 94, 108, 109, 120, 124, 133
Normal modes, 171, 172, 174, 180, 183, 184
- Octave division, 127
Odd/even harmonics ratio, 366, 367
OLA, 238, 244, 251, 254, 265, 274, 280, 282, 405, 407, 408, 430
Oscillator, 246, 247, 404
Oscillator bank, 247, 248, 255, 269, 279, 404, 405
Overdrive, 93, 116–118, 120
Overlap and Add, *see* OLA

- Panorama, 138, 142, 145
- Partial tracking, 477
- Peak continuation, 348, 377, 390, 392, 397, 431, 433
- Peak detection, 383, 384, 390, 431, 433
- Peak filter, 52–54
- Peak measurement, 98, 99, 102
- Peak Program Meter (PPM), 85
- Peak value, 95
- Pentode, 112
- Perceptive features, 336
- Perceptual interface, 176
- Phase, 32, 40, 48, 56, 237, 240, 242, 244, 245, 247, 248, 251, 254, 255, 267–269, 272, 284, 285, 287, 291, 337, 338, 377, 379, 396, 397, 431, 434
 - instantaneous, 338, 404
 - random, 290, 408
 - target, 262, 338, 339
 - unwrapped, 262, 263, 270, 274, 275, 339
 - zero, 287
- Phase difference, 270, 272, 277, 338
- Phase increment, 269, 270, 272, 274, 277, 279, 282
- Phase interpolation, 255, 261, 269
- Phase modulation, 75, 80, 82, 88, 201
- Phase processing, 247
- Phase representation, 254
- Phase response, 27, 39–41, 43, 47, 93
- Phase spectrum, 8, 240
- Phase unwrapping, 255, 261, 275
- Phase vocoder, 238, 242–244, 254, 263, 269, 275, 348
- Phase vocoder basics, 238–244
- Phase vocoder implementations, 244–263
- Phaser, 56–57, 75, 86
- Phasiness, 188, 190, 432
- Phasing, 460
- Phasogram, 244, 259, 260, 289, 292
- Pinna-head-torso, 150, 154, 155, 158, 159
- Pitch, 201–207, 209, 211, 212, 214–216, 221–224, 229, 232, 233, 303, 314, 315, 336, 337, 363, 365, 366, 387, 422
 - discretization, 419
 - transposition, 418, 419, 422
- Pitch detector, 76
- Pitch estimation, 387
- Pitch extraction, 337–360, 367
- Pitch lag, 348–350, 352, 354–358
- Pitch mark, 212, 214, 222–224, 308, 310
- Pitch over time, 347, 348, 360
- Pitch period, 212, 222, 224, 308, 312, 321, 347, 348, 350, 351, 354, 366
- Pitch scaling, *see* Pitch shifting
- Pitch shifter, 217, 220
- Pitch shifting, 126, 147, 201, 202, 215–225, 229, 233, 276–282, 337, 456
- Pitch shifting with formant preservation, 330–336
- Pitch tracking, 336, 337, 343, 347, 360, 477
- Pitch transposer, 217, 220, 221
- Pitch-synchronous Overlap and Add, *see* PSOLA
- Precedence effect, 138, 141, 142, 145, 159, 160, 168, 176, 190, 191
- Prediction error, 304, 305, 307–309, 350–352, 356, 357
- Prediction error filter, 305
- Prediction filter, 304
- Prediction order, 304
- Presence, 175
- Processing
 - block, 6, 24, 27, 29
 - sample-by-sample, 6, 24, 27, 29
- PSOLA, 211–213, 222–225, 227, 231
- PSOLA analysis, 212
- PSOLA pitch shifting, 225
- PSOLA time stretching, 213
- Quantization, 6
- Radiance, 174
- Ray tracing, 173, 185

- Rectification
 full-wave, 126, 127
 half-wave, 126, 127
- Reflection function, 173
- Region attributes, 402
- Release time, 95, 98, 99, 101, 102
- Release time-constant, 84
- Resampling, 201, 217, 218, 222–225, 233, 277, 279, 280, 333, 336
- Residual analysis, 396, 397
- Residual approximation, 397, 399, 407
- Residual component (see also Stochastic component), 379, 391, 396, 400, 401, 403, 407
- Residual synthesis, 407, 408
- Resynthesis, *see* Synthesis
- Reverberation, 137, 144, 145, 149, 152, 169–180, 184, 185
- Rhythm tracking, 477
- Richness of harmonics, 362, 363, 369
- RMS, 84, 85, 89
- RMS measurement, 98, 100
- RMS value, 95, 301, 315–317, 361, 364, 365
- Robotization, 287–289
- Room acoustics, 172, 174–176
- Room presence, 175
- Room-within-the-room model, 149, 160, 167
- Root Mean Square, *see* RMS
- Rotary loudspeaker, 86–88
- Running reverberance, 175, 176, 186
- Sampling, 3, 6
- Sampling frequency, 3
- Sampling interval, 3, 19
- Sampling rate, 3
- Sampling theorem, 3
- Score following, 478
- Segmentation, 402
- Shelving filter, 51–52
- Short-time Fourier transform, 239–241, 243, 375, 376, 379
- Side chain, 95, 130–132
- Sigma Delta Modulation, 499, 501, 502
- ADC, 500
- DAC, 499, 500
- first-order modulator, 504
- linearized model, 504
- second-order modulator, 506
- Signal flow graph, 2, 18, 19, 22, 23
- Single reflection, 143, 144
- Single-side band, *see* SSB modulation
- Sinusoidal analysis, 387, 390
- Sinusoidal component (see also Deterministic component), 377, 400, 401, 403, 416–418, 420, 424
- Sinusoidal model, 376, 383, 391
- Sinusoidal plus residual model, 376, 377, 379, 397, 399
- Sinusoidal subtraction, 379, 396, 397
- Sinusoidal synthesis, 403, 408
- Sinusoidal track, 376, 431
- Slapback, 69
- Slope factor, 97
- SMS, 375, 426, 427, 429
- SOLA, 208–210, 218
- SOLA time stretching, 208, 210
- Sonic perspective, 145
- Sound level meter, 85
- Sound particle, 174
- Sound radiation simulation, 191–192
- Sound trajectories, 147–149
- Soundhack, 158
- Source signal, 300, 301, 312–315
- Source-filter model, 299, 310, 314
- Source-filter representation, 336
- Source-filter separation, 300–315
- Source-filter transformations, 315–336
- Space rendering, 143
- Spaciousness, *see* Spatial impression
- Spatial effects, 137–200
- Spatial impression, 151, 176, 187, 191
- Spatialisateur, 145, 149, 158
- Spatialization, 146, 149, 151, 153, 159, 160, 165, 167–170, 191
- Spectral Centroid, 362
- Spectral centroid, 366, 477
- Spectral correction, 322, 323
- Spectral envelope, 201, 202, 222, 224, 231, 299–304, 307–315, 317, 319, 321–324, 326, 328, 330,

- 331, 333, 334, 336, 350
- Spectral interpolation, 328–330
- Spectral models, 375
- Spectral shape (*see also* Spectral envelope), 401, 402, 418, 420, 422, 428, 429
- Spectral subtraction, 291
- Spectrogram, 15, 57, 59, 244, 259, 260, 289, 292, 342, 343
- Spectrum, 238, 246, 268, 291
 - short-time, 242, 244, 251
 - time-varying, 238, 240
- Spectrum analysis
 - short-time, 15
- Speech recognizer, 426, 428
- SSB modulation, 86, 88
- SSB modulator, 86
- Stability, 21
- Stable/transient components separation, 282–285
- Static function, 95, 97, 99
- Statistical features, 369
- Stautner-Puckette FDN, 180
- Stereo enhancement, 186
- Stereo image, 434
- Stereo panning, 139, 141, 161
- Stochastic component/part (*see also* Residual component), 377, 396
- Subharmonic generation, 126
- Subtractive synthesis, 403
- Sum of sinusoids, 246, 255, 269, 277
- Surround sound, 141, 164
- Sweet spot, 164, 170
- Synchronous Overlap and Add, *see* SOLA
- Synthesis, 237, 238, 242, 244, 247, 255, 263, 269, 270, 272, 277, 288
 - grid, 269
 - hop size, 243, 244, 269, 272, 273, 275, 277, 287
 - window, 244, 274
- Synthesis filter, 303–305, 307–309, 317, 336
- Tangent law, 139, 140
- Tape saturation, 128
- Tapped delay line, 157, 178, 179
- Taylor series expansion, 108
- Threshold, 95, 97, 99, 102, 104, 105, 118, 120
- Timbre preservation, 418
- Time compression, 202, 203, 205, 208, 213, 217, 220
- Time expansion, 202, 203, 205, 208, 213, 217, 220
- Time resolution, 380, 396, 406, 433
- Time scaling, *see* Time stretching, 205, 206, 208, 209, 211, 218, 224, 233, 429
- Time shuffling, 226–229
- Time stretching, 201, 205–214, 217, 218, 222, 223, 225, 229, 268–277, 336, 337, 402
 - adaptive, 368
- Time warping, 440
- Time-constant, 83
 - attack, 84
 - release, 84
- Time-frequency filtering, 263–266
- Time-frequency grid, 243
- Time-frequency processing, 237–297
- Time-frequency representation, 15, 237, 251, 254, 255, 257, 258, 263, 267, 268, 282, 285, 287, 288, 290, 299, 301, 336, 342
- Time-frequency scaling, 268, 269
- Trajectories, 390, 391, 393, 397
- Transaural, 159, 160, 165, 186, 187
- Transaural listening, 165
- Transaural spatialization, 160
- Transaural stereo, 165–167
- Transfer function, 20–22, 24, 26, 34, 36, 39–41, 43, 46, 51, 52, 54
- Transformation, 237, 238, 269, 274
- Transients, 432
- Transparent amplification, 142
- Transposition, 203, 204, 207, 215–217, 228, 322, 326
- Tremolo, 75, 77, 78, 90, 420
- Trill, 460
- Triode, 111–113, 120, 121
- Tube, *see* Valve
- Tube distortion, 122

- Tube simulation, 122, 123
Two-way mismatch, 387, 388

Unit impulse, 18, 26

Valve, 109–112, 114–116, 120, 121, 128
Valve amplifier, 106, 110–113, 115
Valve amplifier circuits, 113
Valve basics, 111
Valve simulation, 109–116
Variable speed replay, 202–206, 217
Varispeed, 203
Vector Base Amplitude Panning (VBAP),
 162
Vector Base Panning (VBP), 162
Vibrato, 68, 75, 82, 86–88, 402, 420,
 429, 460
Virtual room, 149, 159, 167
Virtual sound source, 138–140, 143,
 148, 151, 162, 163, 167–170
Vocoder, 85, *see* Vocoding
Vocoding, 299, 315–321, 478
Voice conversion, 426
Voice/silence, 477
Voice/unvoiced detection, 366
Voiced/unvoiced, 345, 366, 367, 369,
 477
Volterra series expansion, 106
VU-meter, 85

Wah-wah, 55, 75
Warmth, 149, 175
Warping, 322, 326, 328, 440
Waterfall representation, 15, 16, 115,
 116, 119, 120, 122, 124, 126,
 129–132
Wave-Field Synthesis, 164
Waveguide reverberator, 178
Waveshaping, 363, 365, 366
Whisperization, 290–291
Whitening, 319, 321, 322
Window, 240, 244, 251, 252, 254, 258,
 260, 268, 273, 279, 433
 Blackman, 11, 47, 275
 Blackman-Harris, 381, 406, 408
 functions, 10, 11
Gaussian, 275, 310
Hamming, 12, 47, 275, 307, 310
Hanning, 275, 302, 310
rectangular, 265
size, 379, 381, 382
sliding, 238–240, 254
triangular, 407, 408
type, 379, 381, 382
zero-padded, 251, 274
Window length, 242, 290
Window period, 273
Windowing, 251, 252, 257, 258, 274

Z-transform, 21–23, 26, 27, 46
Zero crossing, 126
Zero-padding, 10, 382, 383, 433
Zero-phase, 314, 382