N° ORDRE : 2326

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ BORDEAUX 1

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

par **Sylvain MARCHAND**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Modélisation informatique du son musical (analyse, transformation, synthèse)

Soutenue le : 12 décembre 2000

Après avis de : MM. François Pachet ....(SONY CSL-Paris et LIP6) Rapporteurs
Julius O. Smith III ....(CCRMA, Stanford U.)
Udo Zölzer (U. Federal Armed Forces, Hamburg)

Devant la Commission d'examen formée de :

MM.	André Arnold	Professeur .... et	Président Rapporteur
Mme	Myriam Desainte-Catherine	Maître de conférences, habilitée	Examinateurs
MM.	François Pachet	Maître de conférences, habilité .	
	Robert Strandh	Professeur	
	Horacio Vaggione	Professeur	
	Udo Zölzer	Professeur	

– 2000 –

This document was entirely written using LaTeX.

Foreword

(×2)

Olynth (Greek city destroyed by Philip II, father of Alexander the Great)
Silver tetradrachm dated circa 392 before Christ
Obverse: head of Apollo, god of music
Reverse: lyre with 7 strings

Apollo was said to accompany the choir of the Muses on the lyre.

This 2400-year-old silver coin is one of the very first to represent a musical instrument. Furthermore this coin is not only related to music, but also to sound analysis: In order to check if the metal was really silver, people were used to recognize the timbre of the sound produced when the coin was dropped on a marble slab. That is the reason why silver coins were later called “espèces sonnantes” in French, that is “resonating coins”.

Acknowledgments

First of all, I would like to thank Professor André Arnold who has done me the honor of being the President of the jury of my thesis. His faith in the developing of computer music in Bordeaux was a real encouragement for me.

My warmest thanks go to my thesis director, Mrs Myriam Desainte-Catherine, for her active participation in the present work. She gave indeed constant support to me during this thesis. She put her confidence in me and let me the necessary freedom for my research topics. It was a real pleasure to collaborate with her during these nearly four years of work. Above all, she has made the SCRIME project come true in the meanwhile, thus expanding the French research in computer music and making more things possible.

The SCRIME is an organization for scientific researchers in computer science at the University of Bordeaux and music composers of the Conservatoire to collaborate. I would like to thank all the members of the SCRIME, with special thanks to Professor Christian Eloy who is at the origin of the SCRIME too, and Mr Jean-Michel Rivet who was the very first composer to use my sound model during the compositional process of a piece of his.

I am exceedingly grateful to all the members of the LaBRI, University of Bordeaux 1. In particular, I would like to express my gratitude to Professor Robert Strandh who has transmitted to me his enthusiasm, as well as some of his clever ideas in computer science and skill in English. Special thanks must go to Professor Yves Métivier for his always sound advice and his kind support, especially for my first trip to the University of California at the beginning of my thesis.

I am most grateful to Professor Julius O. Smith III for many reasons. First, he offered me the opportunity to give my very first talk in English during his work-group on digital signal processing at CCRMA, Stanford University. Second, although it was impossible for him to attend the defense of my thesis in Bordeaux, he kindly accepted to review my thesis manuscript and to be a “remote” jury member. Third, his precious advice after the review process has contributed to improve the quality of this document.

Regarding my visit to the universities of California, I would also like to express my gratitude to Professor Chris Chaffe who gave me a warm welcome when I arrived at CCRMA, Stanford University, and to Mr Juan Pampin for our interesting discussions there on functional programming, free software, and the Linux operating system. I am also extremely grateful to Messrs Stephen Travis Pope and Miller Puckette who have welcome me, respectively, at CREATE in Santa Barbara and at the University of California, San Diego. They have introduced to me the researchers and composers of their respective research and creation centers.

I am most grateful to Professor Udo Zölzer for many reasons too. First, his interest in my work since the very beginning and his wise advice are extremely precious for me. Second, he was the first person to reference my work in an article of his. Third, he kindly accepted to review my manuscript and to be a member of the jury of my thesis, and he attended its defense in Bordeaux, although his timetable was very busy.

I would like to sincerely thank two other members of my thesis jury: Dr François Pachet from the SONY Computer Science Laboratory (Paris) and the LIP6 (University of Paris 6) for his support and advice from the very beginning of my thesis as well as for having accepted to review my manuscript, and Professor Horacio Vaggione from the University of Paris VIII for being a visionary in computer music who passed me his taste for multi-scale composition.

My best thoughts also go to Dr Daniel Arfib for his vast knowledge of audio processing and life in general, of great benefit to me, and to Mr André Riotte for all his kindness to me and for having provided me with his original ideas in mathematics on frequency warping.

Sincere thanks go to Mrs Mira Balaban and Michèle Castellengo, together with Messrs Jean-Claude Risset and Trân Quang Haï who, during their visit in Bordeaux, have payed attention to my work and encouraged me to always forge ahead.

My grateful thanks go to the people of the Algory company for their interest in my work, for the rendering of morphing animations presented at the DAFx'99 conference, as well as for the development of the Dolabip project. Thanks also go to Mr Anthony Beurivé for his friendly support throughout my thesis.

Last but not least, thanks go to the members of my family who have supported me and have taken care of almost every material detail on my behalf, thus enabling me to focus on my research subject.

Contents

List of Tables

List of Figures

Introduction

Computer music is by nature a pluridisciplinary research field as it involves not only music and computer science, but also mathematics (arithmetic, trigonometry, etc.), physics (acoustics) or cognitive sciences (psychoacoustics). When computer music aims at transforming sounds, it also deals with signal processing. By definition, signals are functions conveying information. Consequently, signal processing involves both physics (signal) and computer science (processing), since computer science is basically the science for the automatic processing of information.

The present work has been carried out in the context of the SCRIME. The SCRIME is an organization for scientific researchers in computer science at the University and music composers of the Conservatory to collaborate. Projects of the SCRIME should not only be scientifically valid, but also musically relevant. Research projects of this structure are mainly situated in the field of the assistance for composition of electro-acoustic music. We observe and try to understand actual practices of composers of electro-acoustic music in order to provide our research in sound and music modeling with new elements. An important motivation is the study of sound timbre from a perceptual and musical point of view, in close collaboration with psycho-acousticians. Another motivation is to provide composers with tools in harmony with their actual needs.

Sound / Music Dichotomy. Since the very beginning of the research in the field of computer music [Moo90, Roa96], two main trends are developing simultaneously but separately. The first trend deals with symbolic musical structures and is based on the intentions of the composer in order to allow always more sophisticated musical abstractions. This research on symbolic modeling of music is issued from the work of Hiller [HI59] on automatic composition. Musical pieces are defined using atoms (notes) organized into musical structures. However the power of the numerous software tools in this symbolic domain comes up against the boundary separating the note from the sound, since these tools cannot penetrate the opaque sounds in order to efficiently control the micro-structure of the symbolic atoms. The second trend considers computers as instruments for musical sound synthesis [Pie83] rather than tools for assistance for musical composition. While the research related to music mainly concerns musical analysis and composition, the research related to sound deals with sound modeling, analysis, synthesis, and transformation in order to directly manipulate the inner structures of sound. This research on the sound structure is issued from the work of Risset and Mathews on the analysis of musical instrument tones [RM69]. Analysis and synthesis play a central role, even if the expressiveness of the underlying sound model and its musical usefulness should not be left aside in any way.

The sound / music dichotomy appears to result directly from the nature of the paradigms for representing sound: The parameters for their controlling are too far away from the musical parameters involved in the macro-structures of the symbolic level. In order to allow the two parallel trends of research to merge, a sound model should enable a continuity between the atoms of the macro-structures of music (first trend) and the inner representation of sound (second trend).

We propose to focus on the perception of sound rather than its physical cause, in order to unify sound (microscopic) and music (macroscopic). We propose as well to consider the musical intentions of the instrumentalists instead of their physical actions on the instruments. We also propose to try unifying musical writing and sound control in order to enable musical composition on several time scales in a continuous manner, thus allowing the control of musical sounds from the microscopic level (usual musical parameters, timbre) to the macroscopic one (score).

Sound Modeling. Sounds are physical phenomena belonging to the physical world. In order to manipulate digital sounds using a computer we need a sound model, that is a formal representation for audio signals. This model should be as general as possible so that most sounds can be faithfully reproduced and transformed in a natural and musically expressive way. Sound modeling draws the link between the real – analogical – and mathematical – digital – worlds. Extracting parameters for the model from the real world is the analysis stage, while producing a sound from the parameters of the model is the synthesis stage.

Auditory models [ZZ91, Pre00] mimic the functioning of the ear to produce visualizations that are in accordance with the perception. They are closely connected to perception and psychoacoustics. Their main drawback is that they produce representations that are ill-suited for musical transformation or resynthesis of the sound. As a consequence we have to find an acceptable compromise for a sound model well-suited for sound transformation and synthesis while taking perception into account. Apart from auditory models, there are three main families of sound models: physical, abstract, and spectral models.

Physical models [CC98] are related to physics and acoustics. They are modeling acoustic sources, such as real instruments for example. They first put the sound source in equations, then they compute a solution during the synthesis stage. They also deal with the gesture of the performer. Although these models are close to the musical concepts since they take musical gesture into account, they cannot manipulate musical abstractions. Moreover they cannot easily reproduce any kind of existing sounds.

Abstract models – such as the Frequency Modulation (FM) synthesis [Cho73, Moo85b] – propose a mathematical formula for sound. This is often an empirical formula, possibly involving musical parameters. Although they are well-suited for creating new sounds, their main drawback is that they cannot reproduce existing sounds since they lack the possibility of sound analysis.

Spectral models [MQ86, SS90, Ser97b] attempt to parameterize sound at the basilar membrane of the ear. Thus, the resulting sound transformations should be closely linked to the perception. Spectral models based on additive synthesis indeed provide general representations of sound in which many musical transformations can be performed in a very natural and musically expressive way. The next step is to structure the sound model in such a way that musical operations can be simply expressed.

We have conducted musical analysis of electro-acoustic pieces [Del86] in collaboration between scientists and musicians of the SCRIME [DCN00]. They involved well-known musical structures like melody, dynamic, and rhythm. More precisely, we have identified the need for a certain number of manipulations of musical sounds. These manipulations can involve either one sound or several sounds. Among the one-sound transformations are operations on the pitch (transposition, vibrato, etc.), loudness (amplification, tremolo), duration (time-stretching), but also on the timbre (such as filtering for example). Transformations involving several sounds can consist in either interchanging parameters among these sounds (cross-synthesis) or blending parameters among them (morphing), thus providing ways to create hybrid sounds.

Sound Analysis. In order to faithfully imitate or transform existing sounds, spectral models require an analysis method to extract spectral parameters from sounds which were usually recorded in the temporal model, that is audio signal amplitude as a function of time. The accuracy of the analysis method is extremely important since the perceived quality of the resulting spectral sounds depends mainly on it.

There are two main families of analysis methods for spectral models. The first family is issued from short-time Fourier analysis, which produces a series of short-term spectra taken on successive – often overlapping – temporal windows on the original signal. Information about the spectral peaks is then extracted from these short-term spectra in order to provide the model with spectral parameters. A good frequency resolution requires a large window, which leads to a poor precision in time. On the contrary, a good time resolution leads to a poor precision in frequency. This is the well-known trade-off of time versus frequency in the classic short-time Fourier analysis. The second family of analysis methods deals with wavelets. The idea is then to perform a frequency decomposition with a constant quality factor, that is with a frequency precision inversely proportional to frequency. As a consequence, frequency resolution is greater for low frequencies, while time resolution is better for high frequencies. The problem is that the parameters obtained from wavelet analysis are not well-suited for musical sound transformations. Indeed the harmonics of a complex sound must be identified in order to allow musical transformations without dramatically audible artifacts. Thus an important problem with the constant quality factor in the wavelet transforms occurs for high-frequency harmonics, for which the frequency resolution is so bad that several harmonics may be averaged into one coefficient of the wavelet transform.

Spectral models can faithfully reproduce a wide variety of existing sounds provided that an accurate analysis method is able to extract the model parameters from these sounds. Although many analysis methods have been proposed, it turns out that few are accurate enough to suit our needs. Moreover almost none of them have been implemented in open-source software programs. The main interest of an accurate analysis methods, providing precise parameters for the spectral models, is to allow ever deeper musical transformations on sound by minimizing deformations due to analysis artifacts. We are specially interested in reproducing and transforming pseudo-harmonic instrumental sounds [ACM85] as well as the human voice.

Sound Synthesis. Sound models would be useless in most practical applications without an efficient synthesis method being able to generate the audio signal from the model parameters, possibly in real time. Spectral models based on additive synthesis require the computation of a large number of sinusoidal oscillators. The problem is to find a very fast method for generating the sequence of samples for each oscillator with as few operations as possible. This computation could be done using the sine function itself. The resulting synthesis would be accurate but extremely slow. Another possibility is to use the inverse Fourier transform in order to simultaneously generate all the oscillators, provided that the oscillator parameters vary extremely slowly. On the contrary, banks of individual sinusoidal oscillators allow parameter variations and an extremely fine control of each oscillator. Nevertheless these banks of sinusoidal oscillators are mostly used in hardware (VLSI) implementations. We are interested only in hardware-independent methods for real-time synthesis. There is only a small number of algorithms for additive synthesis, and very few are efficient enough to suit our needs. Again, almost none of them have been implemented in practice in open-source programs.

Overview. For short, we aim at designing a sound model both musically expressive and computationally efficient, together with an accurate analysis method as well as an efficient algorithm for real-time synthesis. The remainder of this document is organized as follows.

First of all, the word "sound" has not the same meaning for physicists or musicians. In order to fully understand the objective and subjective aspects of sound, basic knowledge of elements of acoustics and psychoacoustics is required. The aim of Chapter 1 is to provide this basic knowledge. Some notations and elements of mathematics and physics are introduced, such as spectra, partials, noises or transients. Then, short introductions to acoustics and psychoacoustics briefly explain sound emission, propagation, and reception by the human auditory system. Finally, the main musical parameters [Jeh97] perceived by a listener are discussed.

The next three chapters are organized in a similar way for the purposes of homogeneity and clarity. For each of these chapters, we first describe interesting previous works while stressing their main advantages and drawbacks, then we present new results and we make comparisons with the previous works. We also describe practical implementations as free software programs. Finally, we possibly stress some important features or perspectives out of our results at the end of each chapter.

Chapter 2 first presents the well-known temporal model, then focuses on spectral sound models, since they provide general representations for sound well-suited for intuitive and expressive musical transformations. More precisely, we first introduce the phase vocoder and additive synthesis. The models based on additive synthesis have solid mathematical and physical basis but they are extremely difficult to use directly for creating or editing realistic sounds. The reason for this difficulty is the huge number of model parameters – controlling many sinusoidal oscillators – which are physically valid but only remotely related to musical parameters as perceived by a listener. We propose the Structured Additive Synthesis (SAS) model which imposes constraints on the additive parameters, giving birth to structured parameters as close to perception and musical terminology as possible. The SAS model consists of a complete abstraction of sounds according to only four physical parameters, functions closely related to perception. These parameters – amplitude, frequency, color, and warping – are inspired by the work on timbre of researchers like Risset [Ris86a, Ris86b], Wessel [Wes78, Wes79], and McAdams [McA84, MBM99], and by the vocabulary of composers of electro-acoustic music. These model parameters enable to independently modify musical parameters such as pitch, loudness or duration, and constitute as well a solid base for investigating scientific research on the notion of timbre. Since there is a close correspondence between the SAS model parameters and perception, the control of the audio effects gets simplified. Many effects thus become accessible not only to engineers, but also to musicians and composers. But some effects are impossible to achieve in the SAS model. It appears that structuring a model in order to facilitate the design of some kinds of sound transformations gives rise to both restrictions on the sounds that can be represented and impossibilities for other kinds of transformations. In fact structuring the sound representation imposes limitations not only on the sounds that can be represented, but also on the effects that can be performed on these sounds. There is a kind of trade-off of complexity versus feasibility in every sound model. We demonstrate these relations between models and effects for a variety of models from temporal to SAS, going through well-known spectral models. We list their main advantages and drawbacks and focus on the feasibility and the complexity of sound effects in these models. Finally, we present *ProSpect*, our free software architecture for spectral sound manipulation.

Chapter 3 presents some analysis methods for extracting spectral parameters from sounds originally in the temporal model. In this chapter we first present the classic Fourier analysis and we point out its main limitations and imprecisions, then we explain some interesting improvements to this analysis that have been proposed recently. We present our high precision Fourier analysis method using signal derivatives. Precisely, this method extends the classic short-time Fourier transform by also considering the signal derivatives, which effectively leads to efficient spectral parameter extraction. FT^n takes advantage of the first n signal derivatives in order to improve the precision of the Fourier analysis not only in frequency and amplitude but also in time, thus minimizing the problem of the trade-off of time versus frequency in the classic short-time Fourier transform. After the short-time analysis, a partial-tracking phase is necessary in order to reconstruct the evolutions in time of the partials of the sounds. We first describe the most famous partial-tracking strategies, then we explain our way of structuring the partials in order to get the four parameters of the Structured Additive Synthesis (SAS) model. Short-time analysis, partial tracking, as well as SAS structuring are implemented in our *InSpect* analysis program. We have also implemented in *InSpect* an interesting technique for lossless compression of the sinusoidal modeling parameters. Finally, we show how the reanalysis of the parameters coming from initial analysis could turn out to be extremely useful not only to enhance the compression ratio, but also to perform very interesting musical processing on the tremolo or vibrato of the sounds for example. This reanalysis turns out to be of great interest for pitch tracking and source separation too.

Chapter 4 presents some efficient synthesis algorithms. As in Chapter 3, the additive synthesis model plays a central role. Additive synthesis requires the computation of a large number of sinusoidal oscillators. We present the most interesting synthesis methods for additive synthesis and we compare their respective performances. We have implemented the fastest method in our *ReSpect* software tool. We chose to generate each oscillator using the simple recursive description of the "digital resonator" [GS85, SC92]. We explain in details the synthesis algorithm we designed, and especially the way the frequency and amplitude parameters can be changed while avoiding discontinuities and numerical imprecision, then we describe its efficient implementation in *ReSpect*, which has been specially designed for the purposes of real-time spectral synthesis. The synthesis is controlled by a flow of additive parameters with a slow rate. We show how *ReSpect* manages to efficiently up-sample the variations of these parameters using interpolating splines prior to the synthesis itself. We also explain how psychoacoustic considerations such as masking phenomena can help reducing the number of partials, thus speeding up the synthesis process. We present then the synthesis of sounds in our Structured Additive Synthesis (SAS) model, based on additive synthesis. Finally, we explain the way of synthesizing noise as well.

Finally, Chapter 5 presents some of the applications of the Structured Additive Synthesis (SAS) model in the fields of creation and education. These applications are numerous, since the SAS model constitutes a solid base for investigating scientific and musical research on the notion of timbre while favoring the unification between music and sound at a sub-symbolic level [Lem93]. We have experienced that this unification allows migrations of the control among the different levels, from the microscopic (sound) to the macroscopic (music) one, thus enriching the palette of the composer. The SAS model is well-suited for sound exploration and design. One of its advantages is its aptitude for creating hybrid sounds from the combination of several sounds. We also show the use of the SAS model for musical composition. It allows the composers to modify both the micro-structure and the macro-structure of musical pieces in a multi-scale composition, that is to perform musical compositions on several time scales in a continuous manner. A new sound synthesis language for musical composition has been implemented and should provide a way to validate and enrich the model. The SAS model is also of great interest for applications regarding interactive control. A pedagogical tool for early-learning electro-acoustic music is based on this model. It provides sound controls that are well-suited for young children because they are based on sound listening rather than signal synthesis. This tool can also be used for real-time musical performances, and it is in fact the first step to a new visual language for music. Finally, we introduce in-progress applications of the SAS model to singing voice. This model is indeed well-suited for the representation of vowels and allows us to control precisely in time the volume and the pitch as well as the timbre itself.

Chapter 1

Sound Physics and Musical Perception

The fast fluctuations – from tens to thousands per second – of the air pressure at the level of the ears generate an auditory sensation. The word “sound” [BL94] stands for both the physical vibration and the sensation this vibration produces. That is why this word has not the same meaning for physicists or musicians...

In order to fully understand the objective and subjective aspects of sound, basic knowledge of elements of acoustics and psychoacoustics is required. The aim of this chapter is to provide this basic knowledge. First, Section 1.1 introduces some elements of mathematics and physics such as spectra, partials, noises or transients. Section 1.2 gives then a short introduction to acoustics and briefly explains sound emission and propagation, while Section 1.3 presents the reception of sound by the human auditory system. Finally, Section 1.4 lists the main musical parameters perceived by a listener.

1.1 Elements of Mathematics and Physics

The simplest way to represent an audio signal is to consider its amplitude as a function of time $a(t)$, where t is time expressed in seconds. This sound representation as a time-signal is often called the time domain. However another – extremely convenient – way of representing an audio signal is the frequency domain, dealing with both frequencies and amplitudes as functions of time (see [Orf96] for an introduction to this topic).

Regarding complex numbers, we denote by j the imaginary unit, thus $j^2 = -1$. We also use the Euler notation for complex numbers, that is $c = a\ e^{j\phi}$, where a and ϕ are the amplitude (magnitude) and the phase of the complex number c, respectively. We also denote by $\mathrm{Re}(c)$ and $\mathrm{Im}(c)$, respectively, the real and complex parts of the complex number c, so that $c = \mathrm{Re}(c) + j\mathrm{Im}(c)$.

Fourier Transform

The Fourier transform and its inverse transform are mathematical transforms allowing to switch from the time domain to the frequency domain and to do the opposite, respectively. If s and S are the expressions of the same signal in the time and frequency domains, respectively, then the continuous-time Fourier transform and its inverse are given by the following equations:

$$S(\Omega) = \int_{-\infty}^{+\infty} s(t)\ e^{-j\Omega t}\ dt \tag{1.1}$$

$$s(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} S(\Omega)\ e^{+j\Omega t}\ d\Omega \tag{1.2}$$

The radian frequency Ω (in radians per second) is a function of the frequency f (in Hz):

$$\Omega = 2\pi f \tag{1.3}$$

Thus the Fourier transform and its inverse can also be defined using the ordinary frequency:

$$S(f) = \int_{-\infty}^{+\infty} s(t)\ e^{-j2\pi ft}\ dt \tag{1.4}$$

$$s(t) = \int_{-\infty}^{+\infty} S(f)\ e^{+j2\pi ft}\ df \tag{1.5}$$

Discrete-Time Fourier Transform
When s is a discrete-time signal, we note:

$$s[i] = s(i\ T_s) \tag{1.6}$$

The sampling period T_s (in seconds) is the inverse of the sampling frequency F_s. For discrete-time signals, the expressions of the discrete-time Fourier transform (DTFT) and its inverse transform (IDTFT) are, respectively:

$$S(\omega) = \sum_{n=-\infty}^{+\infty} s[n]\ e^{-j\omega n} \tag{1.7}$$

$$s[k] = \frac{1}{2\pi}\int_{-\pi}^{+\pi} S(\omega)\ e^{+j\omega k}\ d\omega \tag{1.8}$$

$S(\omega)$ is a 2π-periodic function. The digital frequency ω is a function of the frequency f:

$$\omega = \frac{2\pi f}{F_s} \tag{1.9}$$

Thus the discrete-time Fourier transform and its inverse can be defined using the frequency:

$$S(f) = \sum_{n=-\infty}^{+\infty} s[n]\ e^{-j2\pi\frac{fn}{F_s}} \tag{1.10}$$

$$s[k] = \frac{1}{F_s}\int_{-F_s/2}^{+F_s/2} S(f)\ e^{+j2\pi\frac{fk}{F_s}}\ df \tag{1.11}$$

And this time $S(f)$ is a F_s-periodic function.

Discrete Fourier Transform

If the discrete-time signal s is of finite length N, then we use the discrete Fourier transform (DFT) and its inverse transform (IDFT), which are special cases of the DTFT and IDTFT, respectively. Let us define:

$$f_k = \frac{kF_s}{N} \tag{1.12}$$

$$\omega_k = \begin{cases} \frac{2\pi f_k}{F_s} \\ \\ \frac{2\pi k}{N} \end{cases} \tag{1.13}$$

The values f_k and ω_k for $k \in [0, N-1]$ are the DFT frequencies in Hz or cycles per sample, respectively. If we note $S[k] = S(\omega_k)$ (for Equations 1.7 and 1.8) or $S[k] = S(f_k)$ (for Equations 1.10 and 1.11), the DFT and IDFT can be defined using the following equations, respectively:

$$S[m] = \sum_{n=0}^{N-1} s[n] e^{-j2\pi\frac{mn}{N}} \tag{1.14}$$

$$s[k] = \frac{1}{N} \sum_{n=0}^{N-1} S[n] e^{+j2\pi\frac{kn}{N}} \tag{1.15}$$

Sometimes a normalization factor $\frac{2}{N}$ is used in Equation 1.14 to read the exact amplitude of sinusoids directly in the spectrum. Then $\frac{1}{2}$ is used instead $\frac{1}{N}$ in Equation 1.15.

1.1.1 Spectrum

Spectra are made of complex numbers. The amplitude (magnitude) and phase spectra corresponding to a complex spectrum S consist of the amplitudes and phases of its complex values, respectively. The magnitude spectrum is also referred to as the power spectrum.

A signal s is real-valued iff its (frequency) spectrum S is conjugate-symmetric (real part is even, imaginary part is odd), as shown in Figure 1.1.

The time and frequency domains are deeply related. Some of these relations are summarized in Table 1.1.

Addition

Addition in the time domain and addition in the frequency domain are equivalent. Moreover a signal s is equal to zero ($\forall t,\ s(t) = 0$) iff its spectrum S is equal zero ($\forall f,\ S(f) = 0$).

Convolution

The convolution sum $*$ is an operation whose definition in the continuous-time and discrete-time cases are, respectively:

$$x(t) * y(t) = \int_{-\infty}^{+\infty} x(u)\ y(u-t)\ du \tag{1.16}$$

$$x[n] * y[n] = \sum_{k=-\infty}^{+\infty} x[k]\ y[n-k] \tag{1.17}$$

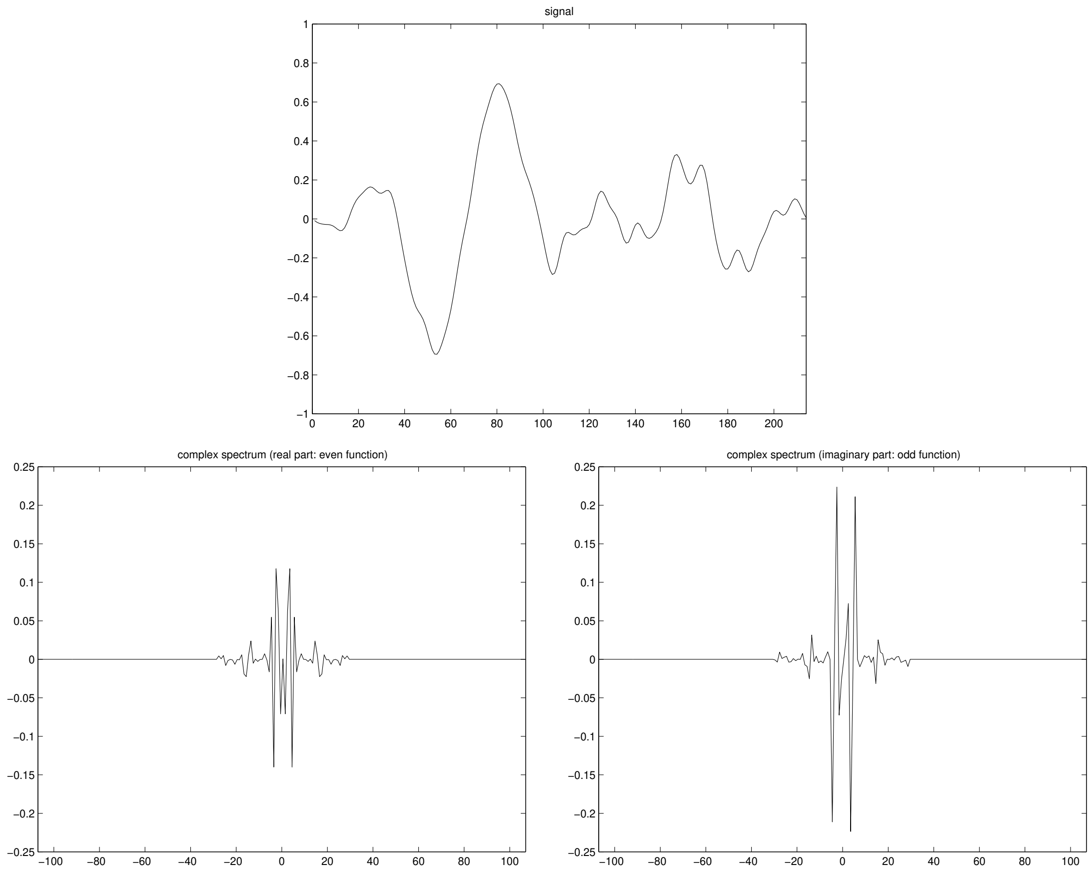


Figure 1.1: The real part (left) and imaginary part (right) of a real-valued signal (top) are even and odd functions, respectively.

time domain	frequency domain
zero 0	zero 0
Dirac impulse	constant
sinusoid	Dirac impulse
addition +	addition +
convolution $*$	multiplication $\times$
multiplication $\times$	convolution $*$
zero-padding	interpolation
interpolation	zero-padding

Table 1.1: Functions (top) and operations (bottom) in the time domain (left) and the corresponding functions or operations in the frequency domain (right).

Convolution in the time domain is equivalent to multiplication in the frequency domain, and multiplication in the time domain is equivalent to convolution in the frequency domain. That is, using the usual uppercase notation for spectra:

$$x * y = X \times Y \tag{1.18}$$
$$x \times y = X * Y \tag{1.19}$$

Zero-Padding

Zero-padding in the time domain consists in increasing the size of a finite-length signal by adding samples with the 0 (zero) value at the end – or at the beginning – of the signal. Zero-padding in the frequency domain consists in artificially increasing the width of the spectrum by adding zeroes for the frequencies that were not in the original spectrum. Zero-padding in the time or frequency domains is equivalent to interpolation in the frequency or time domains, respectively. The interpolation in the frequency domain does not provide a greater resolution in frequency, but only a smoother spectrum.

Sinusoids

A simple Dirac impulse in the time domain has a continuously flat spectrum in the frequency domain, while a sinusoid – which has an infinite support in the time domain – is a Dirac impulse at the corresponding frequency in the frequency domain. Sinusoids play an important role in sound because many sounds are made of a sum of sinusoidal oscillations. A sinusoid is totally described in terms of amplitude, frequency, and phase. Figure 1.2 shows the temporal (time-domain) and spectral (frequency-domain) representations of sounds consisting of superposed sinusoids. These sounds are of the form:

$$s(t) = \sum_{p=1}^{P} a_p \sin(2\pi f_p t + \phi_p) \tag{1.20}$$

where P is the number of sinusoids (*partials*) and the values a_p, f_p, and ϕ_p are the amplitude, frequency, and phase of the p-th partial, respectively. In Figure 1.2, only the magnitude spectra are represented. This corresponds to our perception, since the values of ϕ_p can be changed without changing the perceived sound (see Section 1.3).

1.1.2 Partials

The partials are “peaks” – that is local maxima – corresponding to time-domain sinusoids in the magnitude spectrum (see Figure 1.3). In fact the values a_p and f_p are not constants, but functions of time. In Equation 1.20, the cosine function can be used instead of the sine function, since these functions are indeed the same apart a phase difference of $\pi/2$:

$$\sin(x) = \cos\left(x - \frac{\pi}{2}\right) \tag{1.21}$$
$$\cos(x) = \sin\left(x + \frac{\pi}{2}\right) \tag{1.22}$$

Let us consider the two following signals s_1 and s_2:

$$s_1(t) = \cos(2\pi 5t) \cos\left(2\pi 4000t - \frac{\pi}{2}\right) \tag{1.23}$$

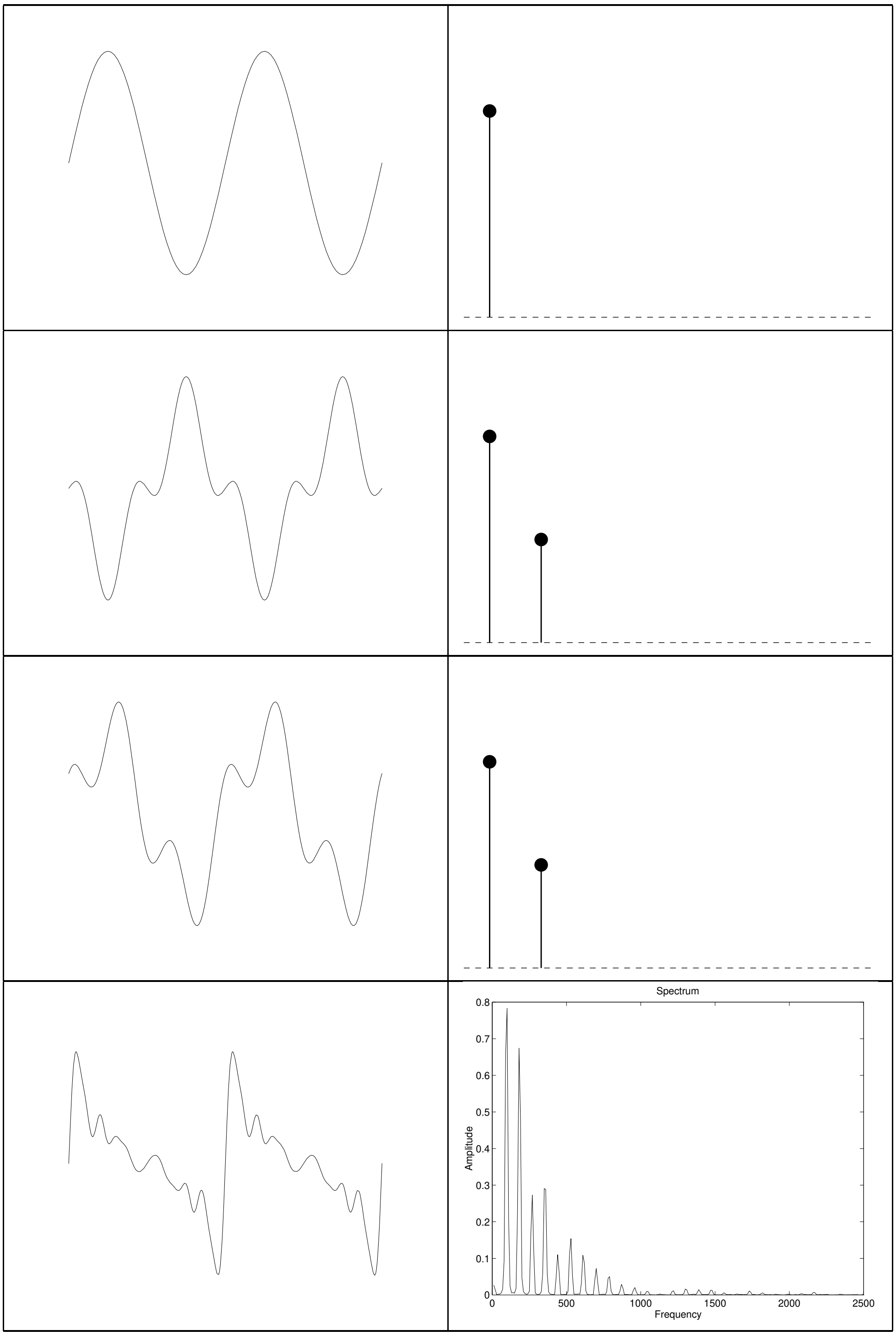


Figure 1.2: Temporal (left) and corresponding spectral (right) representations of sounds consisting of a sum of sinusoidal oscillations. Only the magnitude of the spectrum is represented. From top to bottom are displayed a single sinusoid, the sum of two sinusoids, the sum of the same two sinusoids but with different phases, and a complex sound consisting of many sinusoids. Waveforms corresponding to the same magnitude spectrum by different phase spectra sound the same to the ear. Thus the time-domain representation is not very meaningful.

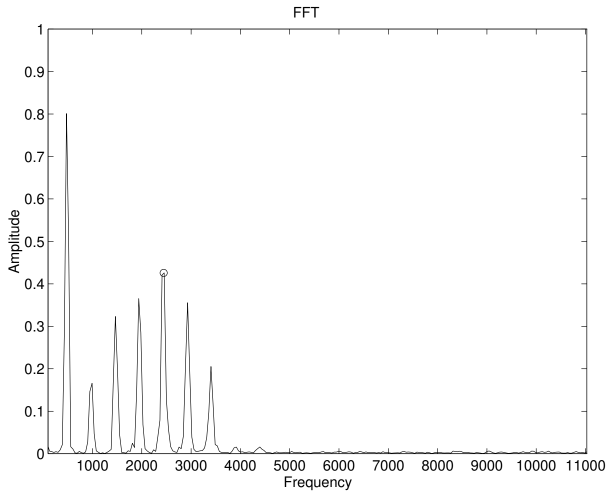


Figure 1.3: Magnitude spectrum resulting from a Fast Fourier Transform (FFT).

that is Equation 1.20 with $P = 1$, $a_1(t) = \cos(2\pi 5t)$, $f_1(t) = 4000$ Hz, $\phi_1 = -\pi/2$, and

$$s_2(t) = \frac{1}{2}\cos\left(2\pi 3995t - \frac{\pi}{2}\right) + \frac{1}{2}\cos\left(2\pi 4005t - \frac{\pi}{2}\right) \tag{1.24}$$

that is Equation 1.20 with $P = 2$, $a_1(t) = 1/2$, $f_1(t) = 3995$ Hz, $\phi_1 = -\pi/2$, $a_2(t) = 1/2$, $f_2(t) = 4005$ Hz, $\phi_2 = -\pi/2$.

In fact these sounds are the same: $s_1 = s_2$, because of the following equations:

$$\cos(p) + \cos(q) = 2\cos\left(\frac{p-q}{2}\right)\cos\left(\frac{p+q}{2}\right) \tag{1.25}$$

$$\cos(a)\cos(b) = \frac{1}{2}\left[\cos(a-b) + \cos(a+b)\right] \tag{1.26}$$

From a strictly mathematical point of view, both formulations are equivalent. But according to perception, Equation 1.23 must be preferred. The ear decides when the mathematics cannot (see Section 1.3).

1.1.3 Noise

Physicists often define noise as an unwanted signal causing interferences on another signal. The musical definition of noise is not clear at all. We will use the word "noise" to designate a random signal resulting from a stochastic process. For "deterministic signals" each value of a sequence is uniquely determined by a mathematical expression or a rule of some type. In many situations the processes that generate signals are so complex as to make precise description of signal extremely difficult or undesirable, if not impossible. In such cases, modeling the signal as a stochastic process is useful. A "stochastic signal" is considered to be a member of an ensemble of signals that is characterized by a set of probability density functions. In other words, for a specific signal at a particular time, the amplitude of the signal sample at that time is assumed to have been determined by an underlying scheme of probabilities.

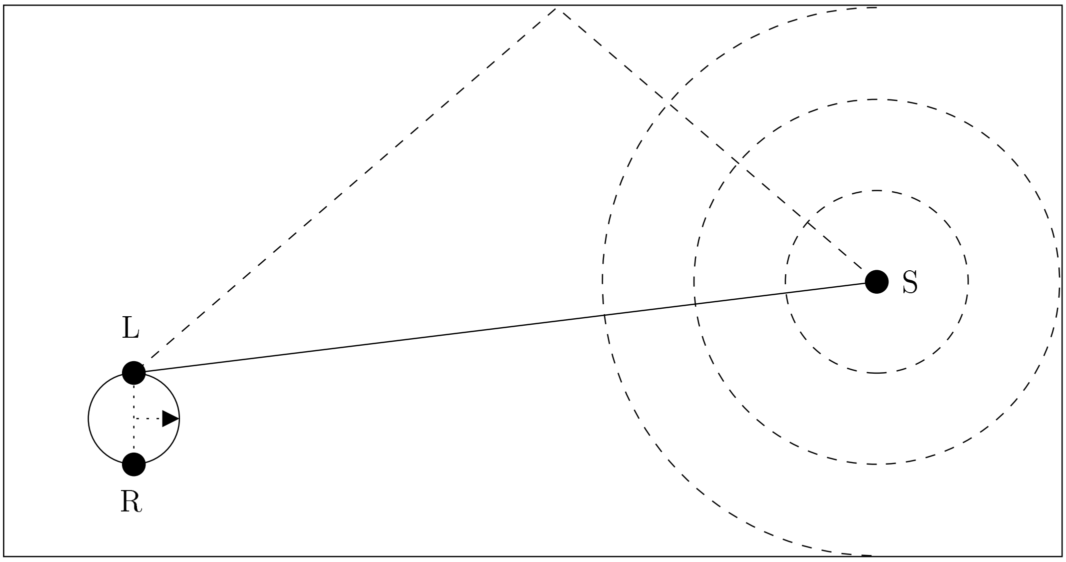


Figure 1.4: Sound propagation in a room. The sound is generated by the acoustic source S, then the wave pressure travels through the space, possibly being reflected by some walls, and finally reaches the left (L) and right (R) ears.

1.1.4 Transients

Transients are very short signals, that are perceived as clicks. Although they can play an important role during the modeling of realistic sounds, we will not study them in details in the remainder of this document.

1.2 Elements of Acoustics

Objectively, sound is a physical phenomenon with a mechanical origin – an acoustic source – producing a local perturbation of the air pressure which propagates into the air, thus giving birth to an acoustic wave.

1.2.1 Sound Sources

To produce sound, something must vibrate. More precisely, a mass located somewhere in space must oscillate around a position of balance. Such an acoustic source can be for example a pipe or a string, excited either by a one-shot excitation or periodically. Castellengo lists in [Cas94] the main families of acoustic sources. If the sound is produced by a series of impulses occurring periodically, the associated spectrum is harmonic and the attack is quite soft. This is the case for the voice. On the contrary, if the sound is produced by an unique impulse, the spectrum is not harmonic anymore and there are transients at the beginning of the sound, followed by a – often long – damping. This is the case for percussive sounds.

1.2.2 Sound Propagation

The sound source produces a local perturbation of the air pressure that propagates into the air, thus giving birth to a wave pressure which is spherical provided that the source is punctual and the air is homogeneous. Figure 1.4 shows an acoustic source S producing such an acoustic wave. For standard conditions the propagation speed is about 330 meters per second. Of course, most sources are directional. For directional sources the intensity of the emitted wave is dependent on the angle in the spherical coordinates centered at S. Anyway, in the remainder of this section we will consider that the source is omnidirectional. For example, let us consider the classic case of a sinusoidal wave of frequency f. Each time the wave hits an obstacle (for example a wall), it changes of direction (reflection) and possibly lose some intensity (absorption), depending on the material constituting the wall. It may also change of phase, possibly with a different amount in function of the frequency f. For diffuse materials the diffraction phenomenon can also occur. Anyway, a tutorial to room acoustics is beyond the scope of this section.

Finally, the wave pressure reaches the receptor – the ear – and things are getting even more complicated...

1.3 Human Auditory System

Subjectively, sound is a sensation rendering the perception [Kit94] by the brain of an event which conveys information from the world. Psychoacoustics is the science that deals with the perception of sound and works on the link between perception and the auditory system. At the center of the human auditory system – more precisely in the inner ear – is the cochlea (see Figures 1.5 and 1.6), which contains the basilar membrane (see Figure 1.7) that performs the frequency decomposition of the sound signals. The sound wave is traveling around the basilar membrane, deforming it. The maximum of the deformation is located on the membrane at a point depending on the frequency of the sound wave. High or low frequencies cause maxima at the entrance or at the back of the cochlea, respectively. In short, the air vibration is transmitted to the ear-drum, amplified, and then analyzed by the cochlea, which sends the electrical information to the brain via the auditive nerve.

Auditory Models. Research is in progress in the field of auditory modeling (see [ZZ91] and [Pre00] for references). Auditory models mimic the functioning of the ear to produce visualizations that are in accordance with the perception. Although they do not take all the mechanisms of the ear yet, they already produce satisfactory results. But there are two main problems. The first problem is that they require a very important amount of computational power for the analysis step. The second one is that they produce representations that are ill-suited for musical transformation or resynthesis of sound.

As a consequence we have to find an acceptable compromise for a sound model well-suited for sound transformation and synthesis while taking perception into account.

1.3.1 Amplitude and Loudness

The Fechner law applies to every sensory organ and claims that the sensation is proportional to the logarithm of the excitation. More precisely, for a sound of intensity I the sound volume

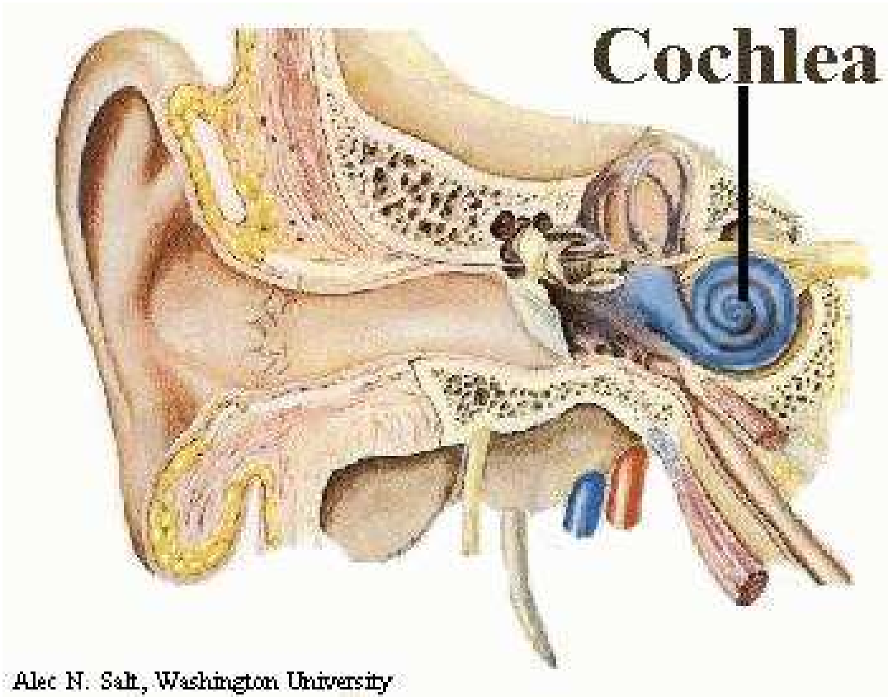


Figure 1.5: Longitudinal section of the right ear. From left to right are the pavilion (pinna), the ear-drum (tympanum), and the cochlea.

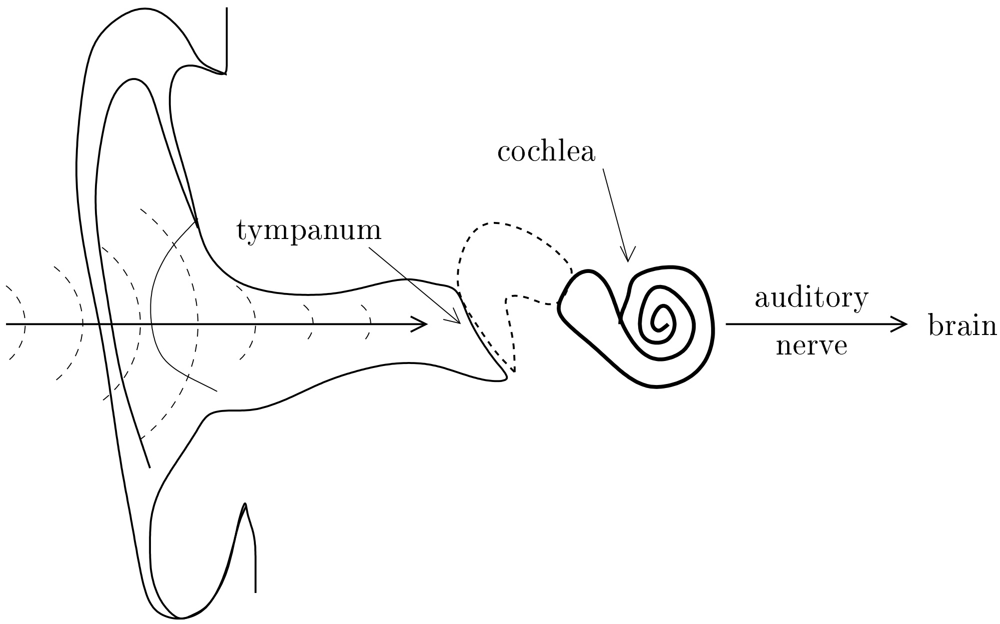


Figure 1.6: Schematic version of the section of the right ear.

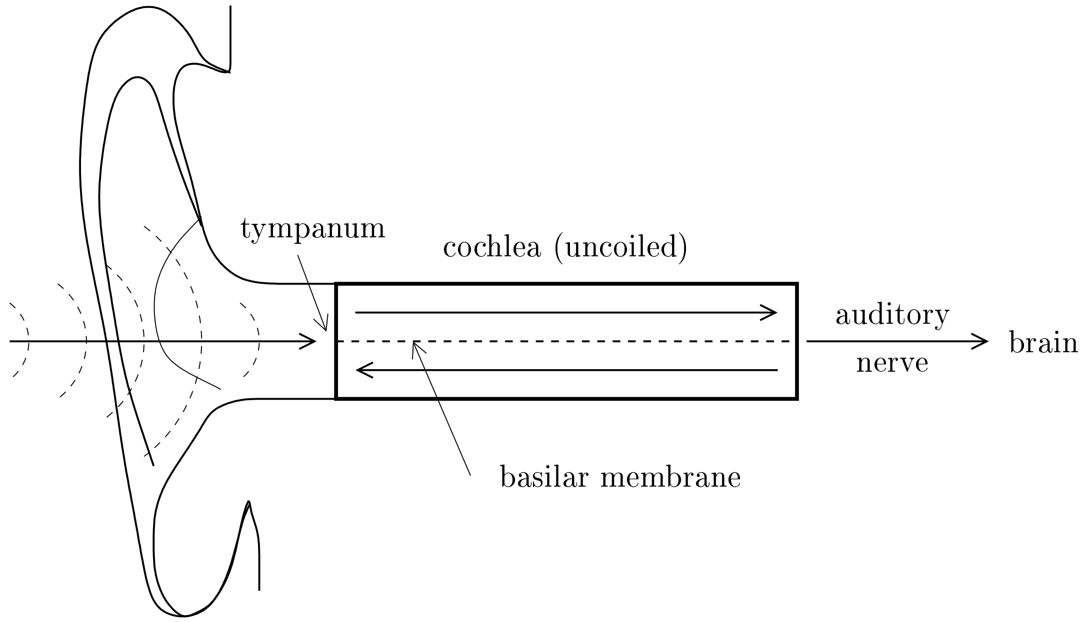


Figure 1.7: Simplified version of the section of the right ear. The cochlea has been uncoiled. Inside the cochlea is the basilar membrane that performs the frequency decomposition of the sound signals.

in dB can be calculated using equation:

$$V(I) = 10\ \log_{10}\left(\frac{I}{I_{0\mathrm{dB}}}\right) \tag{1.27}$$

where $I_{0\mathrm{dB}}$ is the intensity of reference for 0 dB. Since the acoustic intensity I is proportional to the square of the acoustic pressure P, Equation 1.27 can be reformulated in function of the pressure P:

$$V(P) = 20\ \log_{10}\left(\frac{P}{P_{0\mathrm{dB}}}\right) \tag{1.28}$$

where $I_{0\mathrm{dB}}$ is the pressure of reference for 0 dB. Psycho-acousticians take $P_{0\mathrm{dB}} = 2 \cdot 10^{-5}$ Pa (Pascals). But we will consider in the remainder of this chapter the volume as a function of the amplitude of the signal using equation:

$$V(A) = 20\ \log_{10}\left(\frac{A}{A_{0\mathrm{dB}}}\right) \tag{1.29}$$

In fact Equations 1.28 and 1.29 are the same, except that the origin of the volume scale gets simply translated. This is not really a problem since the ear has much more gift for differential analysis than for absolute measurement. As a consequence, only relative values really matter.

Things are not that simple. Indeed the perceptive parameter associated to the volume is in fact loudness, since volume is related to a physical – not perceptive – level. For sinusoids with the same amplitude but different frequencies, the perceived loudness is not the same. In fact loudness is a function of both the amplitude and the frequency of the sinusoid, and this complex relation between amplitude, frequency, and loudness is illustrated by the Fletcher and Munson diagrams which can be found for example in [ZF81]. Human beings can perceive loudness in a range of about 130 dB. In fact this range is not a constant, but a – rather complicated – function of the amplitude and frequency summarized in these Fletcher and Munson diagrams.

However we are interested in modeling complex sounds, not single sinusoids. It turns out that Equation 1.29 together with a range of 130 dB is a good compromise.

1.3.2 Frequency and Pitch

Human beings can hear frequencies in the range of 20 Hz to 22 kHz approximatively, and can identify more than 600 different pitches. More precisely, the precision in the perception of frequency (resolution) is not a constant. Below 500 Hz a difference of $\Delta_f = 1.8$ Hz could hardly be heard. Beyond 500 Hz, we can consider that the imprecision in frequency Δ_f is proportional to the frequency f itself, so that $\Delta_f / f = 0.35\%$. If two frequencies lie in the Δ_f interval then we hear beats and the ear can show a nonlinear behavior. Since the real amplitude of variation of the frequency is twice the imprecision threshold, there are about 140 different pitches below 500 Hz and more than 480 above.

A very convenient scale for representing frequencies is the Bark scale (after Barkhausen), which is very close to our perception [ZZ91]. Equations 1.30 and 1.31 allow to go from the Hertz scale to the Bark scale and to do the opposite, respectively.

$$B(f) = \begin{cases} f/100 & \text{if} \quad f \leq 500 \\ 9 + 4\log_2(f/1000) & \text{if} \quad f > 500 \end{cases} \tag{1.30}$$

$$F(b) = \begin{cases} 100\, b & \text{if} \quad b \leq 5 \\ 1000 \cdot 2^{(b-9)/4} & \text{if} \quad b > 5 \end{cases} \tag{1.31}$$

In fact the Bark is a multiple of a smaller unit called the mel. More precisely, 1 Bark = 100 mels. This is the reason why this scale is also called the melodic scale. For musical applications we should rather consider using the harmonic scale (see Section 1.4).

Again things are no that simple. When the volume increases, the pitch sensation may increase or decrease for high or low frequencies, respectively. For very short sounds, the pitch is lower than the one of longer sounds with the same frequency and timbre. The presence of formants – that is local maxima in the spectral envelope – can modify the pitch sensation, thus leading to "paradoxical" sounds studied by Risset. For example, a formant centered in the high frequencies can lead to a higher pitch.

1.3.3 About the Phase

Is the ear able to perceive variations of the phase, and if the answer is positive, what are the thresholds of this perception? Ohm stated that changes in the phase spectrum, although they altered the wave-shape, did not affect its aural effect [Ris91]. Helmholtz developed a method of harmonic analysis with acoustic resonators [vH54]. According to these studies, the ear is "phase-deaf", and timbre is determined exclusively by the magnitude spectrum. Except for very special cases, we have experienced during the synthesis experiments (see Chapter 4) that Ohm is certainly right, although no one can prove it formally yet. Apart from the practical experiments [CMD+76], there are at least two arguments for believing this, an acoustic argument and a psychoacoustic one.

From the acoustic point of view, just imagine a complex sound going through a room. Each time it hits a wall, its direction changes but also the phases of all of its frequency components. These changes of phase and direction depend on the wall material and on the frequency. While some materials do modify phase on reflection at high frequencies, this does not occur significantly when the wavelength is much larger than the "grain size" of the wall material. Thus, not much of this happens up to a few kHz in most cases. However, when all the reflected sounds finally reach the ear, the phase differences due to the different ray path-lengths are very significant. As a consequence, if phase was important then in theory the perceived sound would be very different depending on whether or not it is the direct sound or one of its reflections for example, whereas in practice the timbre of an instrument does not really change.

The psychoacoustic argument is that it has been measured [ZF81] that the modification of the phases existing between the components of a complex sound can be heard only if two components lie in the same critical band and have sufficient amplitudes. And if at least three components lie in different critical bands (see Table 1.2), it appears that the perception of the phase can be neglected, especially for the perception of the loudness. Of course, for very low-pitched sounds, the phase can produce "beats" in the volume. But this nearly never happens for sound with a "reasonable" pitch, so in the remainder of this chapter we will consider, after Ohm, that the effects of phase can be neglected.

1.3.4 Nonlinearities of the Ear

As previously mentioned, the ear can behave in a nonlinear way. An extremely interesting nonlinear property of the ear is that it integrates certain frequency areas of the audible spectrum in bands called the critical bands.

For every frequency, there is a critical band centered at this frequency. Its width is a function of the frequency itself, corresponding to exactly 1 Bark in the Bark scale. Inside the critical band are audibility thresholds, inversely proportional to the masking threshold which is maximum at the center of the band. Table 1.2 shows that the audible spectrum can arbitrarily be covered with only 24 critical bands.

Masking Phenomenon. The existence of critical bands results in a masking phenomenon [ZF81, ZF90, ZZ91]. Physically, the addition of two signals of the same amplitude is ruled by a nonlinear addition law and gives a maximum of +6 dB. However, from a perceptive point of view, there is a modification of the perception threshold for a sound m (masked sound) when it is played together with a louder sound M (masking sound). Consider the case of M and m being two sinusoids of frequency f_M and f_m, respectively [WL24]. In first approximation, the masking threshold looks like a triangle in the Bark-dB scale, as shown in Figure 1.8. If f_m is close to f_M, the sound m is masked by the sound M and becomes inaudible. This phenomenon can be used to lower the number of sinusoids to be computed during additive synthesis (see Chapter 4).

Of course, as always, things are a bit more complicated. Beats can be perceived when $f_m \approx nf_M$ (n being a positive integer). Moreover the two sounds can interact to create perceptive frequencies at $if_m \pm jf_M$ (i and j being integers), thus creating "virtual" harmonics for a pure sinusoid. It can lead to the addition of a "virtual" perceptive fundamental when the fundamental is physically missing.

Number	Lower Boundary (Hz)	Central Frequency (Hz)	Upper Boundary (Hz)
1	20	50	100
2	100	150	200
3	200	250	300
4	300	350	400
5	400	450	510
6	510	570	630
7	630	700	770
8	770	840	920
9	920	1000	1080
10	1080	1170	1270
11	1270	1370	1480
12	1480	1600	1720
13	1720	1850	2000
14	2000	2150	2320
15	2320	2500	2700
16	2700	2900	3150
17	3150	3400	3700
18	3700	4000	4400
19	4400	4800	5300
20	5300	5800	6400
21	6400	7000	7700
22	7700	8500	9500
23	9500	10500	12000
24	12000	13500	15500

Table 1.2: Only 24 critical bands are sufficient to cover the whole audible spectrum.

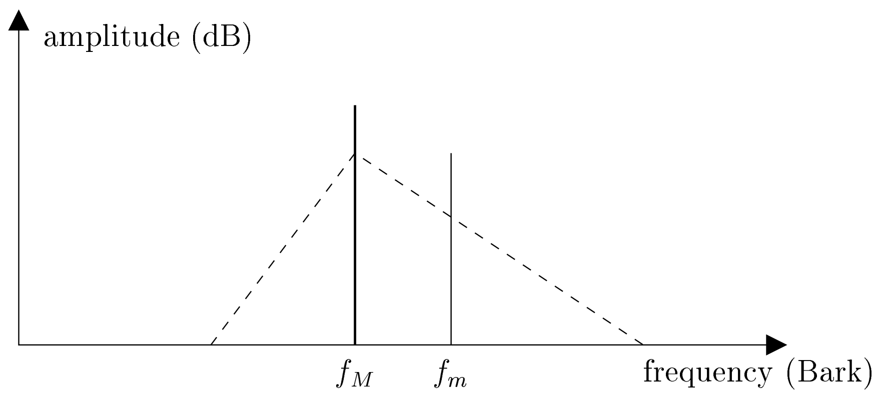


Figure 1.8: Masking of a sinusoid of frequency f_m by another sinusoid of frequency f_M. The masking effect is maximal when f_m and f_M are close. As a first approximation we can consider that the masking threshold is close to a triangle in the Bark-dB scale, although it is not exactly the case in practice, especially for the top of the triangle.

Another interesting phenomenon is temporal masking. There are two kinds of temporal masking. The post-masking occurs when the masking sound disappears. In fact, its effect persists in time during some milliseconds. As a consequence, even if the masking sound is not present the masking effect is still present, although it decreases with time. Perhaps more surprisingly, pre-masking also exists. More precisely, the masking effect is active a few milliseconds before the masking sound really appears. However this phenomenon is less important.

1.4 Musical Parameters

Sound has a cultural and artistic dimension too. While the occidental world focuses mainly on the duration, volume, and pitch of the sound and try to notate these parameters, the oriental world is more oriented towards timbre and the music is not written.

Performed music is a continuous stream of sound. Notational systems are by nature discrete, so capturing music in a notation first requires that the elements to be rendered are parsed out of the sound stream. A good example of this is the quantization of the continuous musical stream into notes. Notes are event specifications that code pitch, onset time, and duration.

In the traditional music notation, notes are indicated as large dots (the note head), with vertical stems placed on a longitudinal grid of five lines (the staff). The shape of the head and the presence or absence of flags on the stem, or beams connecting stems, determine duration. Multiple-note heads can be grouped on a stem, producing a chord. Notes are sometimes connected to curved lines called slurs to show their grouping into phrases. The flow of time is indicated on the staff by the accumulated durations of the notes. Height on the staff determine pitch. Notes of the same pitch connected by slurs are said to be tied, meaning that the duration of their group is the compound of slurred notes. This notation is extremely complicated.

The traditional music notation bears little resemblance to a formal language. The degree of abstraction is such that is telescopes many levels of description together, resulting in a system that is extremely concise and effective for humans, if formally rather ambiguous. Apart from this ambiguity, the main problem is that where a continuum is quantized, information is necessarily lost. The trick is then that the information not carried in the score can be recovered through a set of implicit rules. Thus a satisfactory realization of an encoded work can be reconstituted through the interpretive practice of trained performers. A performer can use the rules of musical interpretation to reconstitute an acceptable stream of sound.

In the traditional music notation only duration and pitch are systematically notated, and loudness less so. Translating even the apparently straightforward dimensions into physical parameters of frequency and amplitude is more difficult that it first appears. Perceived pitch is, of course, strongly correlated with the frequency of a periodic – or nearly periodic – signal, but it can also depend on amplitude and timbre.

In order to define sound models for computers, a formal representation of sound is needed. The first attempt is to formalize the note. A note has an onset time, a duration, and an amplitude envelope that is a function of time. This envelope is often simplified, and decomposed into four phases: attack, decay, sustain, and release (see Figure 1.9). Although this simplification is often well-suited for representing the macroscopic variations of the amplitude, it cannot represent tremolo (sinusoidal variations) for example. We propose to generalize this representation. In this model, the note has an onset time and a duration, an amplitude envelope that

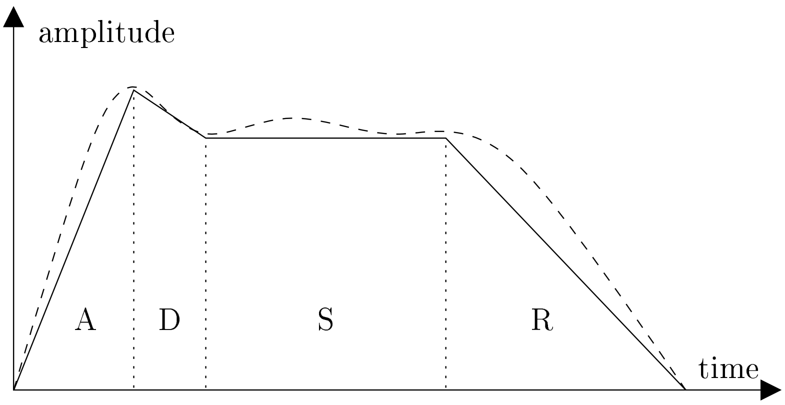


Figure 1.9: The simplified amplitude envelope of a note, composed of the attack (A), decay (D), sustain (S), and release (R) phases.

is a continuous function of time, as well as a frequency envelope that may vary over time too. This representation is close to the physical and perceptive parameters and is convenient for any kind of sounds, whether coming from the occidental or the oriental traditions. The problem is now to represent the timbre.

The principal perceptual dimensions in music are indeed the complex psychoacoustic domains of rhythm, loudness, pitch, and timbre.

1.4.1 Volume

Although loudness is the real perceptive parameter, we will use the volume as an approximation. Equations 1.32 and 1.33 give the relations between the amplitude A and the volume in the dB scale:

$$V(A) = 20 \log_{10}\left(\frac{A}{A_{0\mathrm{dB}}}\right) \tag{1.32}$$

$$A(V) = A_{0\mathrm{dB}}\ 10^{V/(20\mathrm{dB})} \tag{1.33}$$

The variations of the amplitude over time constitute the dynamic of the music. When the variation is a sinusoid with a frequency around 10 Hz, the musical effect is a tremolo. When this variation is monotonous (mathematically speaking), then depending on whether it is increasing or decreasing, the musical effect is either a fade-in or a fade-out.

1.4.2 Pitch

We will use the harmonic scale for the pitch. In fact, the melodic scale is the real perceptive scale (see Section 1.3). Although the harmonic and melodic scales do not coincide, these two scales are approximately the same (apart from a translation) for frequencies below 1 kHz. Since most musical sounds have a fundamental frequency below 1 kHz, this is not really a problem in practice. However perceptive incoherences may occur for high-pitched sounds.

Recall that pitch is not a physical parameter, but a perceptive one. There is a close link with frequency, but this relation is complex. For a single sinusoid, Equations 1.34 and 1.35

give the relations between the frequency F and the pitch in the harmonic scale P:

$$P(F) = P_{\text{ref}} + O \log_2\left(\frac{F}{F_{\text{ref}}}\right) \quad (1.34)$$

$$F(P) = F_{\text{ref}}\, 2^{(P-P_{\text{ref}})/O} \quad (1.35)$$

where P_{ref} and F_{ref} are, respectively, the pitch and the corresponding frequency of a tone of reference. In the remainder of this chapter we will use the values $P_{\text{ref}} = 69$ and $F_{\text{ref}} = 440$ Hz. The constant O is the division of the octave. An usual value is $O = 12$, leading to the classic dodecaphonic musical scale. With these values, P is the MIDI pitch [MMA96], where 69 corresponds to the `A3` note, 70 to `A#3`, etc. There is a lower limit in our perception of the pitch [PPK99]. For an harmonic sound, the perceived pitch corresponds to a kind of greatest common divisor (*gcd*) of the frequencies of the harmonics [Ram65], that is the fundamental. The fundamental corresponds to the frequency of the first harmonic. But this first harmonic may be missing, or "virtual". For a narrow-band noise, the pitch corresponds to the frequency of the middle of the band. For a rippled noise, the pitch corresponds to the *gcd* of the peaks in the spectral envelope, even if the first peak is missing.

Pitch is a very complicated parameter. The helix is a good mental representation for pitch (see Figure 1.10). The height of a tone is seen as one-dimensional property, whereas the chroma is seen as a two-dimensional property [Rév44]. Consider a sequence of rising sevenths on this structure. The tone chroma produces actually a descending scale of seconds. This effect can be demonstrated with Shepard tones [She82, She83].

The macroscopic variations of the frequency over time constitute the melody of the music. When the variation is a sinusoid with a frequency around 10 Hz, the musical effect is a vibrato. When this variation is monotonous (again mathematically speaking), then the musical effect is a glissando or a portamento.

1.4.3 Timbre

By definition, timbre is what allows us to differentiate two sounds with the same loudness and pitch. However this definition is not clear. Anyway, if two parameters of a sound model correspond, respectively, to the loudness and the pitch, then the remaining parameters constitute the timbre. Of course spectral components plays an important role.

Any sound requires a certain amount of time before it is possible to qualify its timbre. There is no unique value, but there is an agreement on saying that human beings need at least 50 milliseconds before recognizing the timbre of a sound. The ear behaves like a temporal window which is sliding in time while looking at the evolutions of the spectral components.

Regarding the perception, it appears that the timbre is dissociated from the pitch in our auditory short-term memory [Dem00, SD91, SD93]. Another very important feature is that musicians often characterize timbre simply as "tone color", for example "sharp", "dull", "bright".

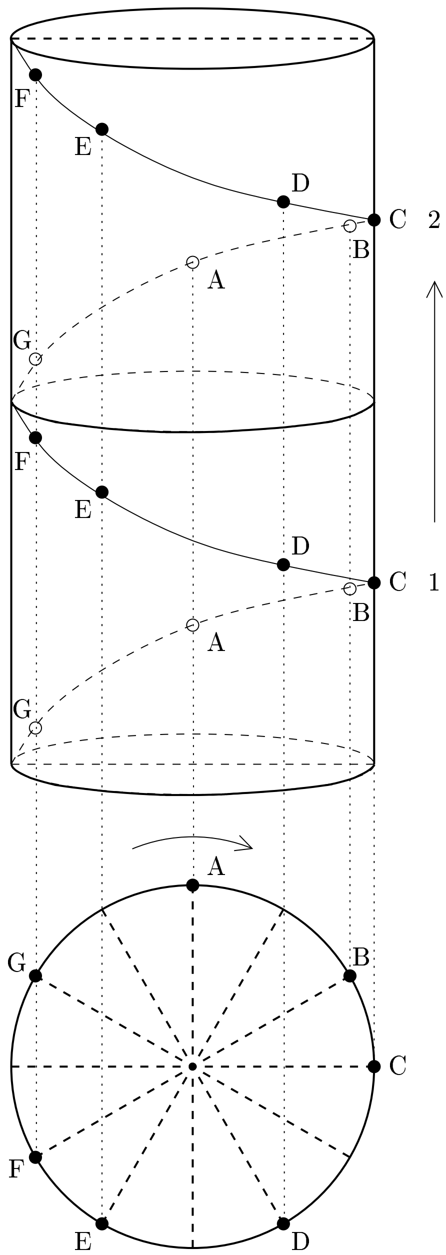


Figure 1.10: The helix as a mental representation for pitch. The height of a tone is seen as a one-dimensional property, whereas the chroma is seen as a two-dimensional property.

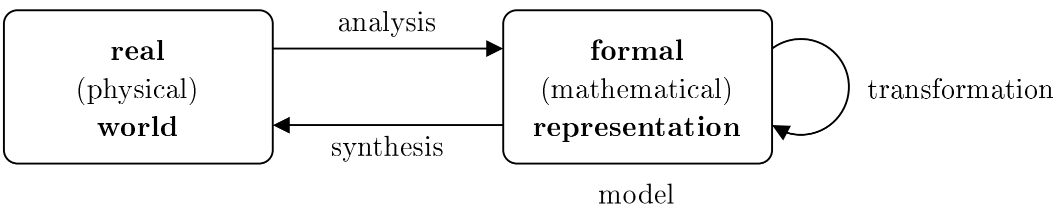
Chapter 2

Sound Models and Musical Transformations

Sounds are physical phenomena belonging to the real world. In order to synthesize new digital sounds or manipulate existing ones using a computer, we need a formal representation for audio signals. A sound model constitutes such a mathematical representation. It should be as general as possible so that most sounds can be faithfully reproduced and transformed in a natural and musically expressive way. Sound modeling draws the link between the real – analogical – and mathematical – digital – worlds. Extracting parameters for the model from the real world is the analysis stage, while producing a sound from the parameters of the model is the synthesis stage.

There are three main families of sound models: physical, abstract, and spectral models. Physical models (see [CC98]) first put the sound source in equations, then compute a solution during the synthesis stage. They are not signal models, because they are modeling the acoustic sources – for example the instruments – instead of the audio signals themselves. Abstract models – such as the Frequency Modulation (FM) synthesis [Cho73, Moo85b] – propose a mathematical formula for sound. This is often an empirical formula, possibly involving musical parameters. Spectral models parameterize sounds at the human receptor. Since they attempt to parameterize sound at the basilar membrane of the ear, the resulting sound transformations are closely linked to the perception. Thus, the design of such sound effects should be more musically intuitive. Spectral models indeed provide general representations of sound in which many musical transformations can be performed in a very natural and musically expressive way. The next step is to structure the sound model in such a way that musical operations can be simply expressed.

The Structured Additive Synthesis (SAS) [DCM99b, DCM99a, Mar99, Mar00d] imposes constraints on the additive parameters, giving birth to structured parameters. Since there is a close correspondence between the SAS model parameters and perception, the control of the



audio effects gets simplified. Many effects thus become accessible not only to engineers, but also to musicians and composers.

Several analyses of electro-acoustic pieces have been performed in collaboration between scientists and musicians. We have identified the need for a certain number of manipulations of sound, that we have determined to be straightforward in our model. Among these are modulation, mixing, filtering, time-stretching, cross-synthesis, morphing, as well as new ways to create hybrid sounds.

But some effects are impossible to achieve in the SAS model. It appears that structuring a model in order to facilitate the design of some kinds of sound transformations gives rise to both restrictions on the sounds that can be represented and impossibilities for other kinds of transformations.

In fact structuring the sound representation imposes limitations not only on the sounds that can be represented, but also on the effects that can be performed on these sounds. There is a kind of trade-off of complexity versus feasibility in every sound model. We demonstrate these relations between models and effects for a variety of models from temporal to SAS, going through well-known spectral models. We list their main advantages and drawbacks and focus on the feasibility and the complexity of sound effects in these models.

Section 2.1 presents the well-known temporal model, while the remainder of this chapter focuses on spectral sound models, since they provide general representations for sound well-suited for intuitive and expressive musical transformations according to perception. Sections 2.2, 2.3, and 2.4 introduce the phase vocoder, additive synthesis, and structured additive synthesis models, respectively. Section 2.5 describes then the design and control of the main musical sound transformations in the SAS model and explains the way to create hybrid sounds from several sounds expressed in this model. It also lists some of the effects which are impossible to implement in the SAS model. Finally, Section 2.6 presents *ProSpect*, our free software architecture for spectral sound manipulation.

2.1 Temporal Model

The simplest way to represent an audio signal is to consider its amplitude as a function of time $a(t)$, where t is time expressed in seconds. This sound representation as a time-signal is often called the time domain or the temporal model. Figure 2.1 shows a sound with this representation. In this model $a(t)$ represents the acoustic pressure at one point in space. The stream of samples of the discrete version of a can be sent directly to the digital-to-analog converter (reconstructor) of any sound card and, to obtain this stream of samples from real sounds, hardware analog-to-digital converters (samplers) exist too. Sampling and reconstruction correspond, respectively, to the analysis and synthesis stages for the temporal model.

2.1.1 Sampling

Let a be a real-valued signal, band-limited in frequency. This means that a has spectrum in some interval $[-\Omega_a, +\Omega_a]$, which is the case iff all the coefficients of its Fourier transform A corresponding to parts of the frequency domain outside this interval are zero. More formally:

$$\exists \Omega_a > 0, \ \text{support}(A) \subseteq [-\Omega_a, +\Omega_a] \text{ where } A(\Omega) = \int_{-\infty}^{+\infty} a(t)\ e^{-j\Omega t}\ dt \tag{2.1}$$

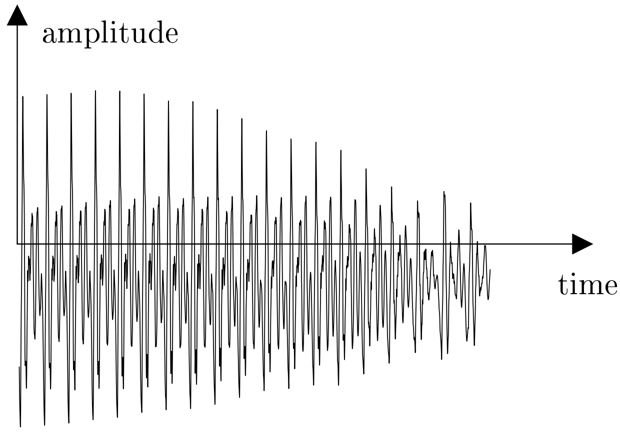


Figure 2.1: A sound represented in the temporal model.

Uniform Sampling

Uniform sampling consists in sampling a using a T_s-periodic impulse train, to obtain a discrete signal where $a[n] = a(nT_s)$. Figure 2.2 illustrates this. Since the Fourier transform of an impulse train is another impulse train, the multiplication of a with the impulse train in the time domain is equivalent to the convolution of the other impulse train with the $[-\Omega_a, +\Omega_a]$ spectrum in the frequency domain. It results in periodically repeated copies of the Fourier transform of a shifted by integer multiples of the sampling frequency $\Omega_s = 2\pi/T_s$ (in radians per second), superposed to produce the spectrum of the impulse train of samples. These shifted copies of the $[-\Omega_a, +\Omega_a]$ interval should not overlap, or else aliasing occurs. To avoid this phenomenon, the condition $(\Omega_s - \Omega_a) > \Omega_a$ must be satisfied, that is $\Omega_s > 2\Omega_a$. This is the well-known Nyquist criterion of the Shannon-Nyquist theorem [Nyq28, Sha49] (and $2\Omega_a$ is the Nyquist frequency). As a consequence, to avoid aliasing, a sinusoid of frequency F Hz cannot be sampled with less than $2F$ samples per second.

Down-Sampling a Discrete Signal. Uniform sampling in the time domain is equivalent to periodization in the frequency domain. This is still true for discrete signals. The same considerations must be taken into account when performing down-sampling. If a is now a real-valued vector of finite length N, the discrete version of Equation 2.1 is:

$$\exists M < N/2,\ A[m] = 0 \text{ for } M < |m| < N/2 \text{ where } A[k] = \sum_{n=0}^{N-1} a[n] e^{-j2\pi\frac{kn}{N}} \tag{2.2}$$

The periodization constant being exactly N/T_s for a given sampling period T_s, it turns out that the sampling period should satisfy the Nyquist criterion $T_s \leq N/M$. For example, in Figure 2.3 the spectrum of the down-sampled signal by a factor 2 coincides with the 2-periodized version of the spectrum of the original signal.

Irregular Sampling

Irregular – that is non-uniform – sampling is also possible, resulting in a discrete signal a_s with $a_s[n] = a(t_n)$, where the $(t_n)_n$ sequence is increasing but does not follow an arithmetic

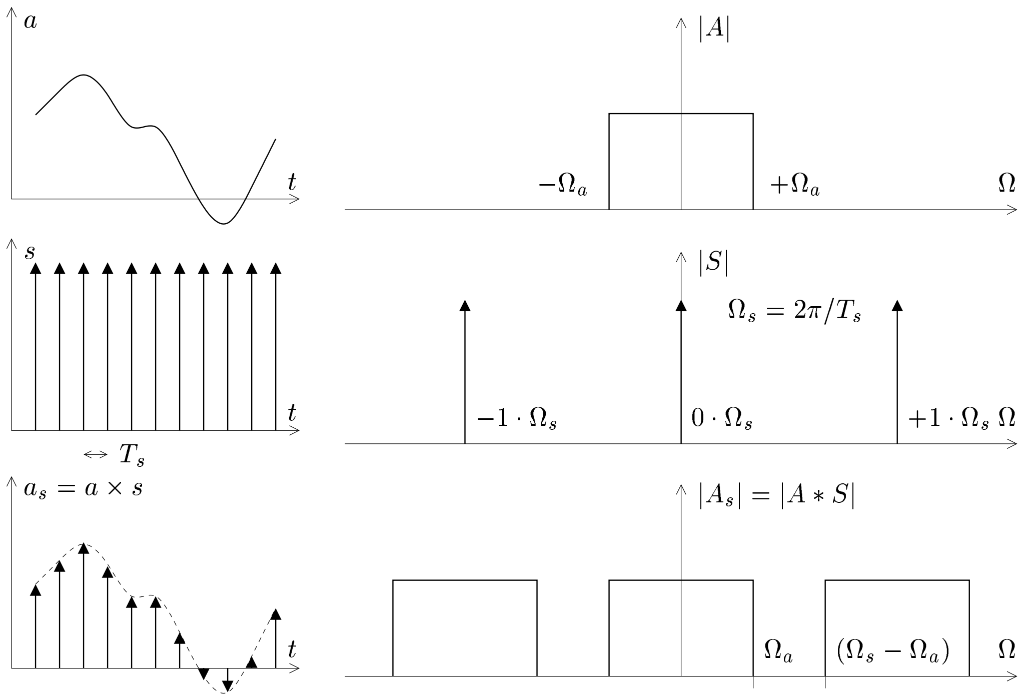


Figure 2.2: Uniform sampling with sampling period T_s. The time-domain and frequency-domain representations are displayed on the left and on the right, respectively. From top to bottom are the continuous-time signal a, the sampling signal s (which is a T_s-periodic impulse train), and the sampled signal a_s.

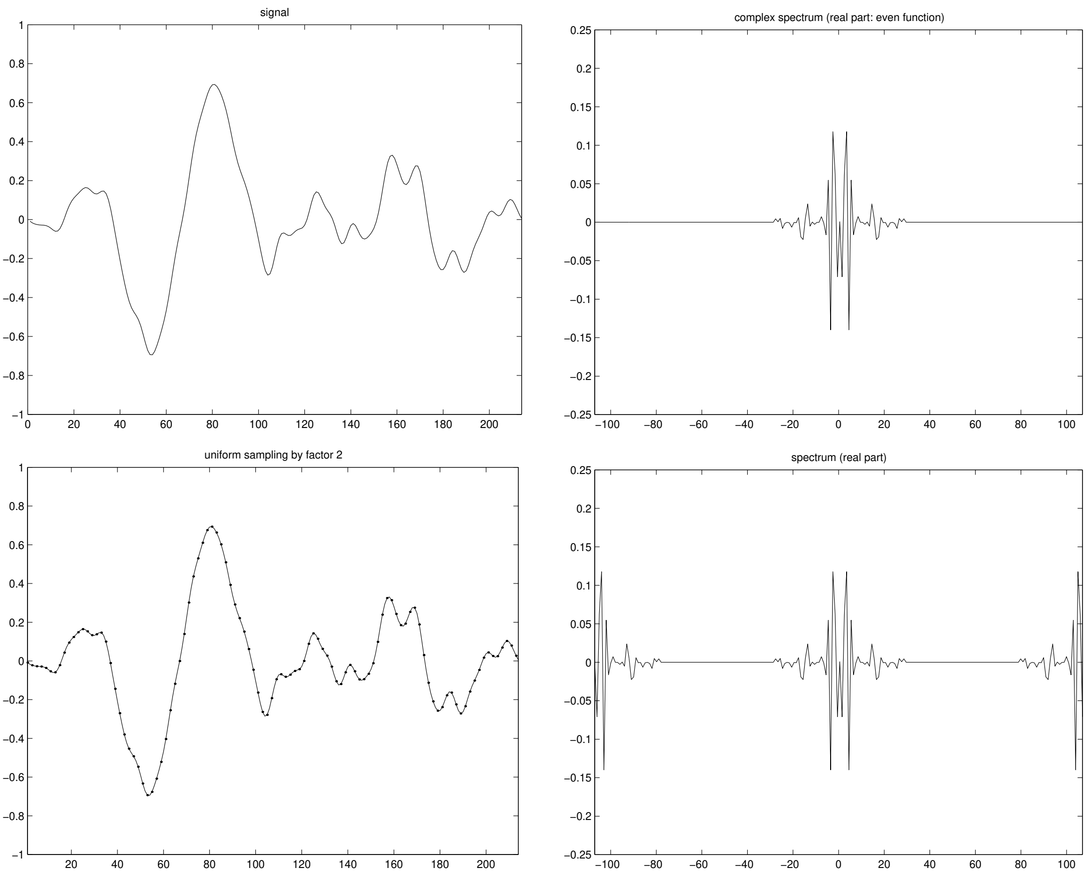


Figure 2.3: Down-sampling by a factor 2 (bottom) corresponds to a 2-periodization of the initial spectrum (top).

progression. It is also possible to reconstruct the original signal a if the maximal distance between two consecutive sampling times t_n does not exceed the Nyquist period.

2.1.2 Reconstruction

Any band-limited signal a with spectrum in $[-\Omega_a, +\Omega_a]$ can then be completely reconstructed from its samples, provided that the Nyquist criterion has been respected during the sampling. However the methods for reconstructing sampled signals are different for the uniform and irregular cases.

From Uniform Sampling

Reconstructing the continuous version of the signal a from its discrete values resulting from uniform sampling is quite easy. In fact a simple low-pass filter will do it, provided that its cutoff frequency is in the $[\Omega_a, \Omega_s - \Omega_a[$ interval. Figure 2.4 illustrates this. The sampled signal a_s is multiplied with the impulse response r of the low-pass filter in the time domain to obtain the reconstructed signal a. In fact, to reduce transients at the beginning and at the end, the signal is mirrored:

$$\begin{cases} s[-i] & = & 2\ s[0] - s[i] \\ s[(N-1)+i] & = & 2\ s[N-1] - s[(N-1)-i] \\ & \text{for} & 0 < i < N \end{cases}$$

This extrapolation by mirroring the signal ensures the continuity of the signal and of its first derivative at the beginning and at the end, and provided that r is symmetric the values of the reconstructed signal match the values of the sampled signal at these positions. The problem is that the theoretical (ideal) reconstruction filter is a box in the frequency domain, corresponding to a *sinc* function in the time domain, more precisely to $\mathrm{sinc}(F_s t)$ where:

$$\mathrm{sinc}(t) = \frac{\sin(\pi t)}{\pi t} \quad (\text{and } \mathrm{sinc}(0) = 1) \tag{2.3}$$

Unfortunately this function has an infinite support. For practical use, it has to be truncated. To avoid aliasing phenomena, one may tapper the truncated version of the *sinc* function by multiplying it with a bell-shaped window. We use the Hann window, but other windows can be used instead (see Chapter 3). Smith uses Kaiser windows in [SG84, Smi00], because its low-pass characteristics can be adjusted. Anyway the resulting function is always the impulse response of a low-pass filter with a finite impulse response and looks like a "Mexican hat" (see Figure 2.5).

Up-Sampling a Discrete Signal. In the same way, complete reconstruction of a discrete signal a from its periodized spectrum resulting from down-sampling is possible provided that the shifted copies of the spectrum do not overlap. The reconstruction is again a simple low-pass filtering operation. Figure 2.6 illustrates the reconstruction of a discrete signal previously down-sampled by a factor 2.

From Irregular Sampling

Reconstructing the continuous version of the signal s from its discrete samples taken in an irregular way is also possible, but a little more complicated. Indeed the uniform recovery

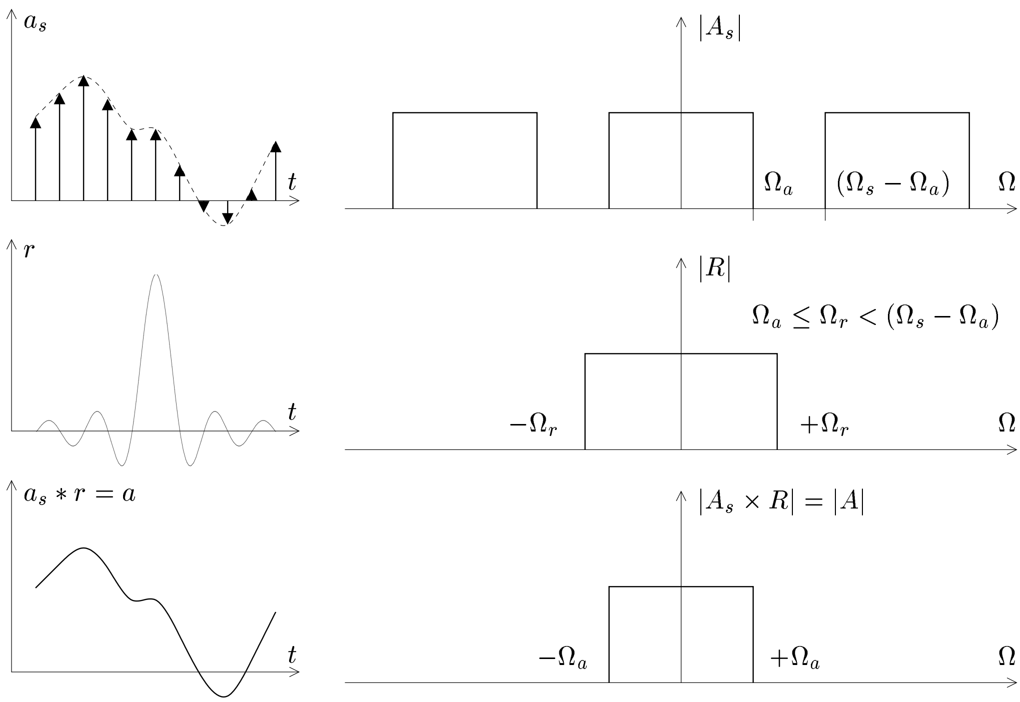


Figure 2.4: Uniform reconstruction of the sampled signal of Figure 2.2. The time-domain and frequency-domain representations are displayed on the left and on the right, respectively. From top to bottom are the discrete signal a_s, the reconstruction signal r (which is a *sinc* function, here translated and truncated for the display), and the reconstructed signal a.

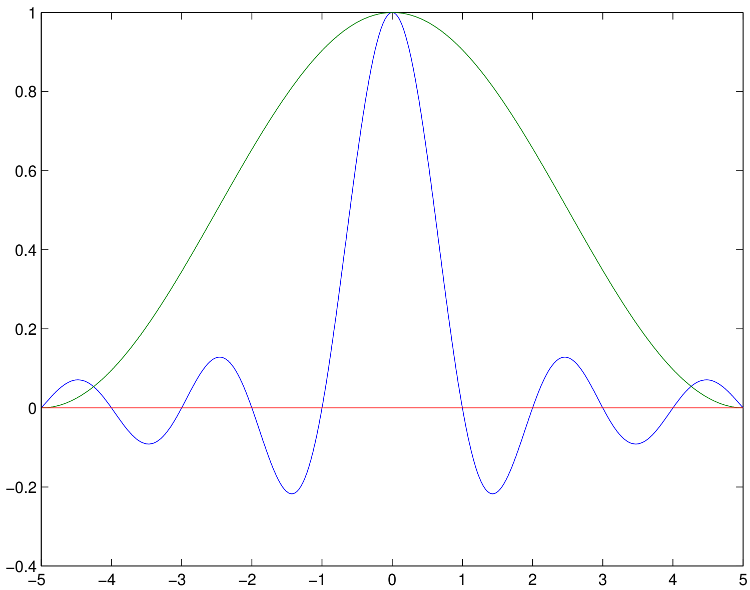


Figure 2.5: Practical reconstruction filters are often *sinc* functions multiplied by bell-shaped windows (the Hann window here), thus looking like "Mexican hats".

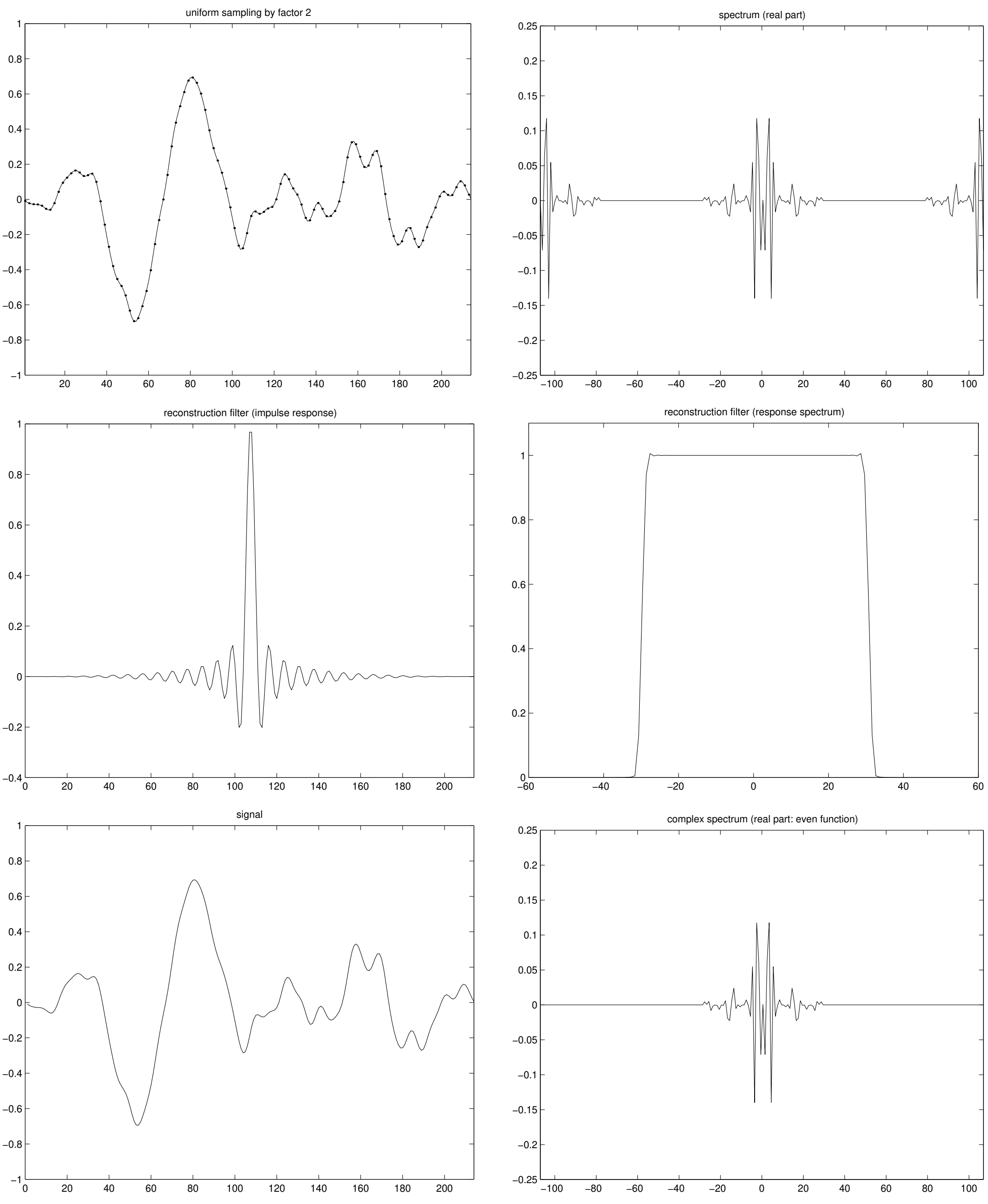


Figure 2.6: Up-sampling – that is reconstruction – of the down-sampled signal of Figure 2.3, using low-pass filtering.

method – consisting of a simple low-pass filtering operation – fails in the case of irregular sampling, as shown in Figure 2.7. Most irregular reconstruction algorithms are iterative in nature [FCS91]. Starting from some initial guess, typically based on the given sampling values, further approximations of a are obtained step by step, using the available (assumed) knowledge about Ω_a.

This is the case of the Allebach algorithm, which is made of 3 steps. Step 1 consists of the interpolation of the sampling values. The interpolated signal contains many high frequencies outside of $[-\Omega_a, +\Omega_a]$. The information concerning Ω_a can be used next. In step 2 the interpolated signal is low-pass filtered with a cutoff frequency slightly greater than Ω_a. Let a_1 denote the first signal resulting from steps 1 and 2, then look at the difference signal $a - a_1$. According to its construction, a_1 has its spectrum within the same range $[-\Omega_a, +\Omega_a]$, and for obvious reasons we know its coordinates at the given sampling positions. Therefore, the estimate indicated above can be applied. Step 3 is the recursive reconstruction of the error – if significant – so that we can again recover a certain portion of the remaining signal by repeating the first two steps, now starting with the sampled coordinates of $a - a_1$. Continuing to use the difference between the given sampling values of a and those of the n-th approximation we generate additive corrections which lead stepwise to improved approximations. Figures 2.8, 2.9, and 2.10 illustrate the Voronoi-Allebach algorithm, that is the Allebach algorithm with Voronoi interpolation as step 1. The Voronoi interpolation corresponds to the nearest neighborhood interpolation, using the arithmetic mean for equally-spaced neighbors. In other words, given the sampling values at known positions we form a step function which is constant from midpoint to midpoint of the given sampling sequence. Because of the assumed smoothness of the signal, this step function will not be too far away from the original signal, in the mean-squared sense. With the Voronoi interpolation, the convergence of the Allebach algorithm is fast. The filtering process destroys the point-wise interpolation property of the Voronoi method, but theoretical considerations show that if the maximal gap is smaller than the Nyquist period then the iterative scheme will converge to the original signal a at a geometric rate (the faster the smaller the actual maximal gap is compared to the Nyquist rate). Figure 2.11 shows the reconstructed signal a_{10} after 10 iterations of the Voronoi-Allebach algorithm. The problem is that the computation of the Voronoi interpolation for multidimensional signals is slow. Other interpolations can be used in step 1 of the Allebach algorithm. For example, the Marvasti method consists in using the trivial interpolation (all the unknown values are set equal to zero) instead of the Voronoi interpolation. The problem is that the convergence of the Allebach algorithm is then much slower.

Strohmer has studied in [Str93, FS92] irregular reconstruction for multidimensional signals. For (one-dimensional) sounds or (two-dimensional) surfaces, the ACT [FGS95] and ABC [Str97] algorithms can be used, respectively. ACT stands for Adaptive weights [FS93] + Conjugate gradient + Toeplitz system while ABC stands for Adaptive weights + Block Toeplitz matrices + Conjugate gradient. Toeplitz systems provide clever linear algebra to save computation time. The conjugate gradient algorithm is a fast and iterative way of computing the inverse of a Toeplitz matrix, to solve Toeplitz systems. The adaptive weights approximation consists in multiplying every sampling point with a weight factor depending on the distance to its neighbor sampling points. It is computationally cost-less and speeds up the convergence of the iterative reconstruction process.

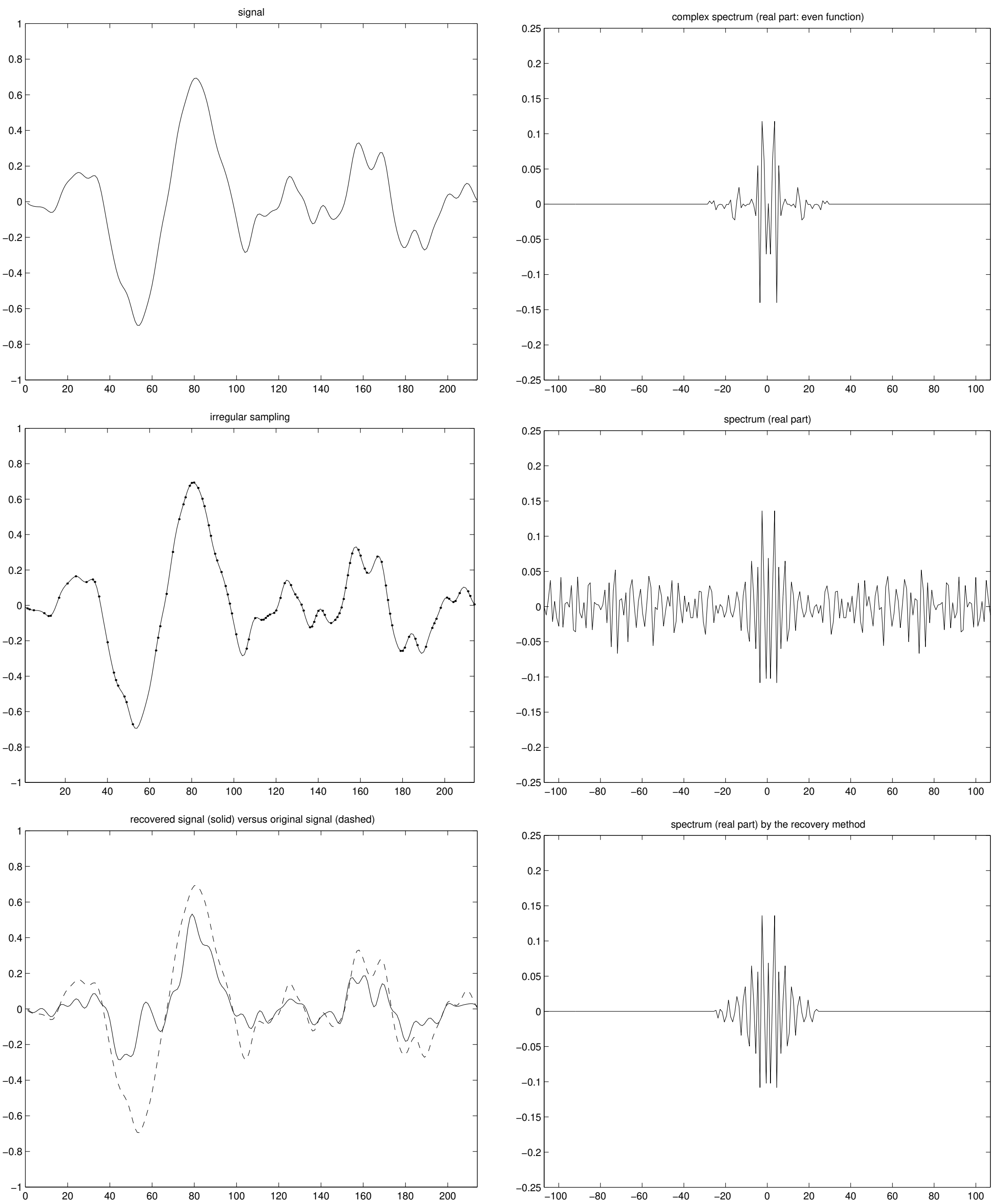


Figure 2.7: The uniform recovery method (low-pass filtering) fails in the case of irregular sampling.

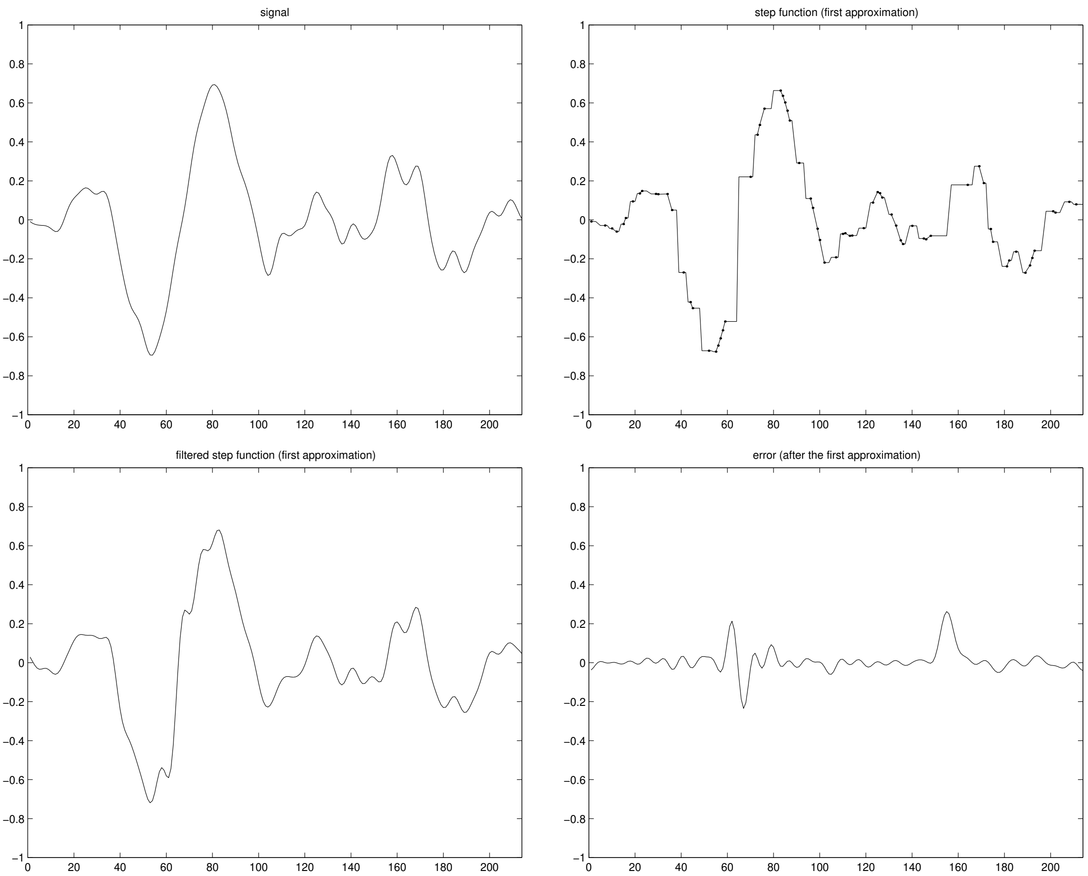


Figure 2.8: The first iteration of the Voronoi-Allebach algorithm. The Voronoi interpolation produces a step function which is low-pass filtered, then the reconstruction error is measured (for known sampling points) and reconstructed, recursively. The reconstruction error converges to 0 if the Nyquist criterion was respected during the sampling stage.

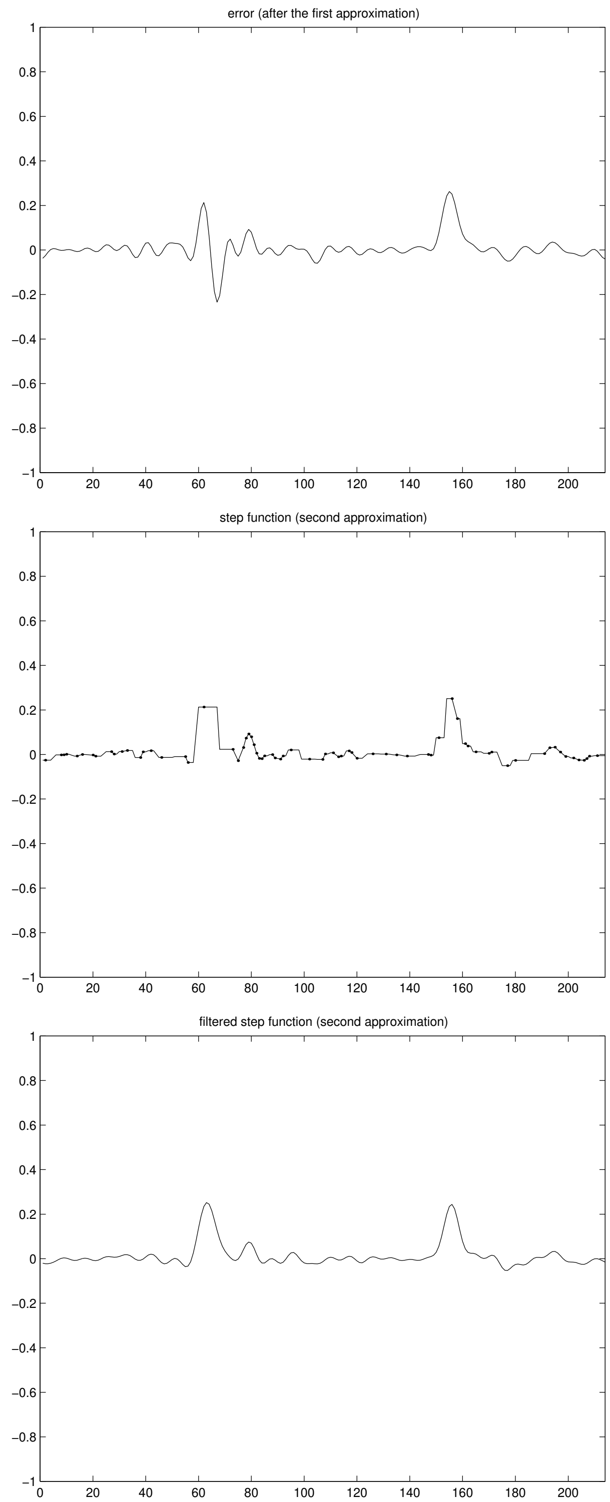


Figure 2.9: The second iteration of the Voronoi-Allebach algorithm.

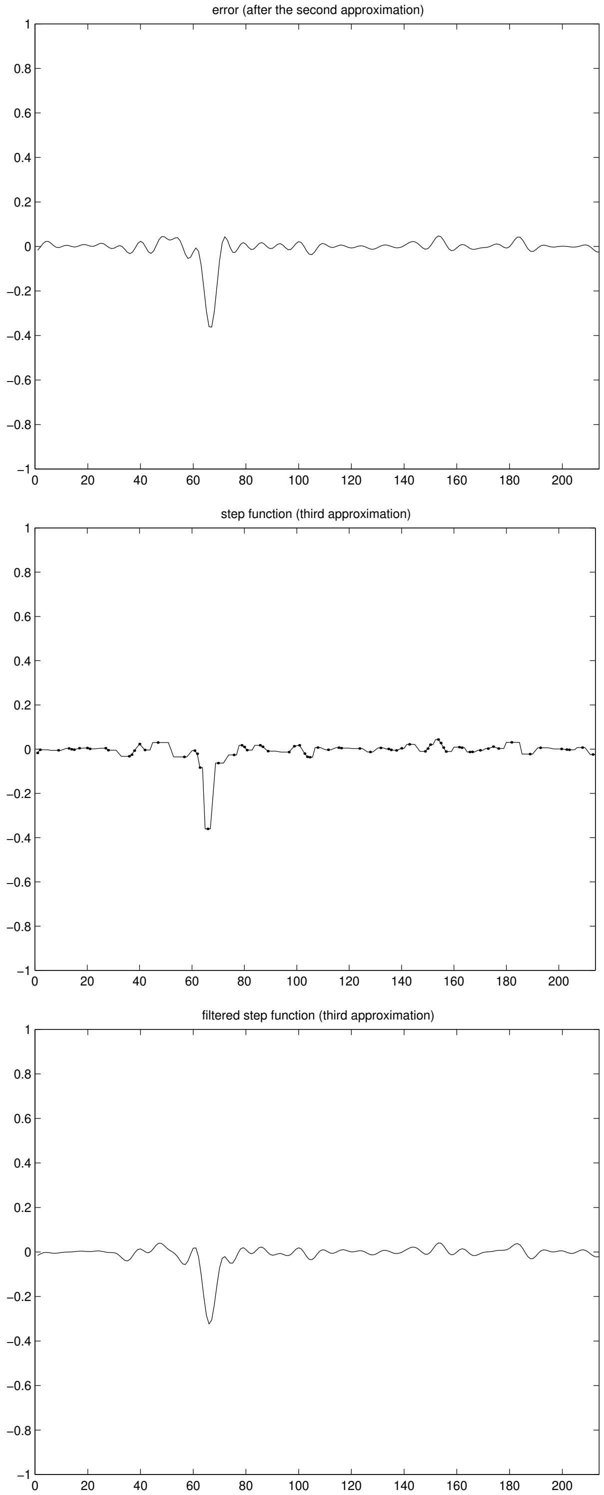


Figure 2.10: The third iteration of the Voronoi-Allebach algorithm.

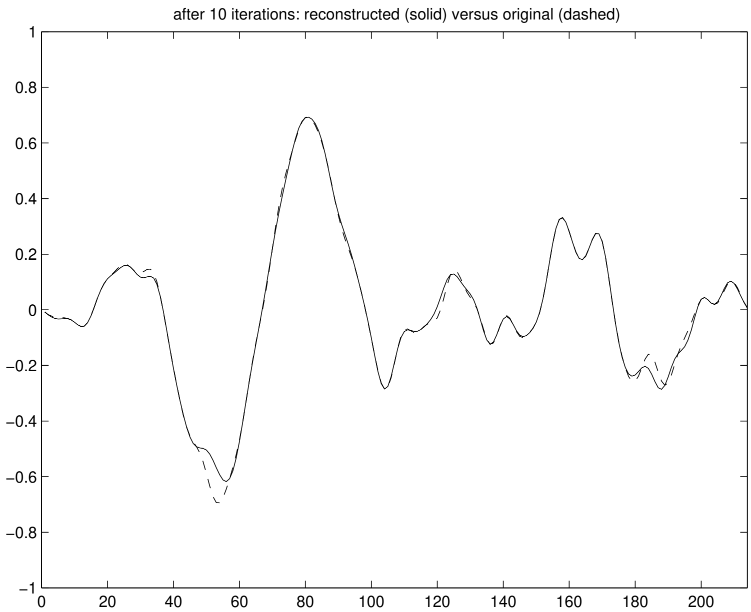


Figure 2.11: Reconstructed signal after 10 iterations of the Voronoi-Allebach algorithm.

parameter	Amplitude	Frequency	Timbre	Time
Amplitude	×			
Frequency		×	×	×
Time		×	×	×

Table 2.1: Interdependence of the musical parameters in the temporal model. When changing a certain parameter (rows), other parameters may change too (columns).

2.1.3 Transformations

All band-limited sounds can be represented in the temporal model. The main advantage of this representation it that everything is possible: There is no restriction on the sounds that can be reproduced or on the transformations that can be performed.

Its main drawback is probably that none of these transformations are really musically intuitive, and it is well known that the simplicity of this model has severe drawbacks as soon as the inner structures of the sound have to be manipulated.

Of course amplification can be performed simply by multiplying the signal with an amplification factor, provided that it varies “slowly”, or else amplitude modulation occurs, thus changing the sound timbre. But pitch transposition while preserving duration is a real challenge [BJ95], and time-stretching without pitch-shifting is another one [Fed98].

In fact the only musical parameters that can easily be modified are the volume (by scaling the amplitude axis) and the pitch (by scaling the time axis), and this last transformation also modifies the sound duration, tempo, and timbre in an unnatural way because it also shifts its formants. In fact the musical parameters are interdependent, as shown in Table 2.1.

The temporal model is however well-suited for effects like echo or reverberation, as they involve a superposition of pressure waves replicas in time. When the impulse response of the room is known, reverberation is similar to filtering. Filtering a sound requires either the composition of a filtering function with a, or convolving a with the impulse response of the filter, or even switching to the frequency domain and multiplying their respective spectra.

Of course deeper transformations can also be performed in the time domain, but their mathematical foundations are not so intuitive and many require to switch to the frequency domain by the mean of a Fourier transform...

2.2 Phase Vocoder

The Fourier transform converts the temporal signal (amplitude versus time) into a spectral representation (amplitude versus frequency). It gives the frequency image of the whole sound, and the entire signal is averaged into a single spectrum. This spectrum coincides with our perception only for stationary sounds. Because most sounds evolve in time, sound models must have time-varying parameters.

The phase vocoder [Moo78c, Dol86, Ser97a] uses the short-time Fourier transform [All77, Por80]. This time-dependent version of the Fourier transform goes through the entire sound while producing a series of short-term spectra taken on successive – often overlapping – temporal frames, which are small pieces of temporal signal. In practice this is a discrete signal resulting from uniform sampling. Then, for each frame window x of N consecutive samples, a (discrete) windowing function w is applied (multiplied) to x, and a (discrete) Fourier transform X is computed:

$$X[m] = \frac{2}{N} \sum_{n=0}^{N-1} w[n]\ x[n]\ e^{-j\,\frac{2\pi}{N}nm} \tag{2.4}$$

The result is a (short-term) spectrum as shown in Figure 2.12. The $\frac{2}{N}$ factor in Equation 2.4 was added for the purposes of normalization. In this model all band-limited sounds can be represented, there is no restrictions on the transformations that can be performed on the sounds, and there is a fast algorithm – the Fast Fourier Transform (FFT) [CT65] – to go from the temporal model to this model, and the inverse transform to do the opposite. In fact this is not a real sound model, but mainly a way to provide a spectral representation for sounds in the temporal domain.

The problem is that there are many physical parameters that are not musically relevant such as the type, size, and hop of the analysis window w, and these parameters have a great impact on the quality of the analyzed sounds. The main advantage is that the sounds are now represented in a spectral domain. As a consequence, filtering gets easier: It is just a matter of multiplication among spectra once the spectral response of the filter is designed, which is quite easy to do. In fact this is not that simple, since artifacts may occur. Time stretching remains quite difficult [SRD, DSR87], mostly because the phases of the short-term spectra must be changed with great care. Again because of the phases, it is extremely difficult to edit these time-varying short-term spectra in a musical way [Str87, Car95].

2.3 Additive Synthesis

Additive synthesis (see [Moo77]) is the original spectrum modeling technique. It is rooted in Fourier's theorem, which states that any periodic function can be modeled as a sum of sinusoids at various amplitudes and harmonic frequencies. For pseudo-periodic sounds, these

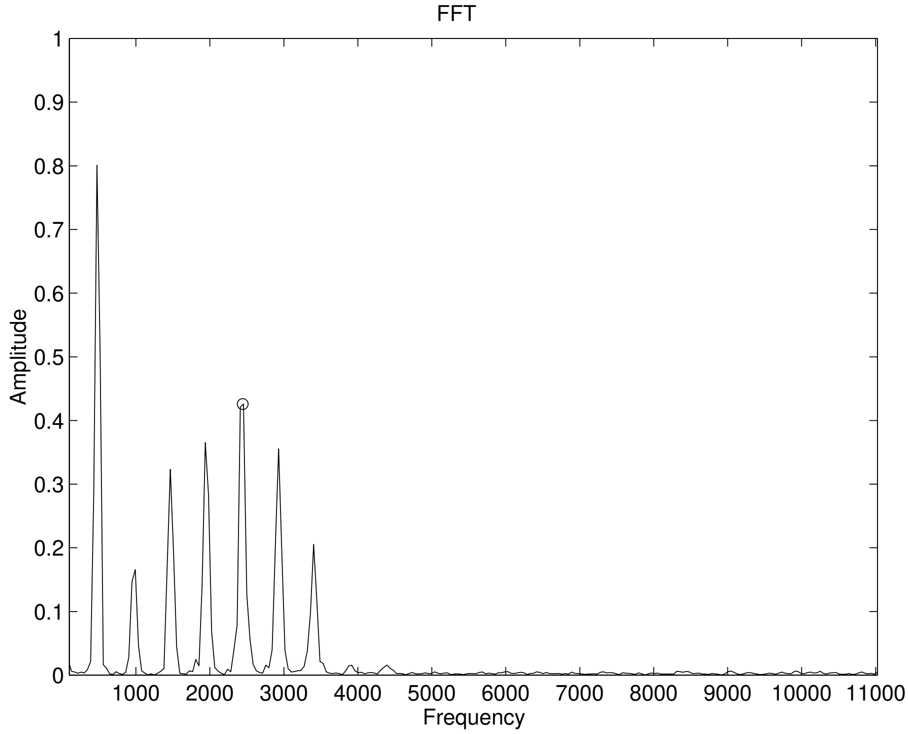


Figure 2.12: Magnitude spectrum resulting from a Fast Fourier Transform (FFT).

amplitudes and frequencies evolve slowly with time, controlling a set of pseudo-sinusoidal oscillators commonly called *partials* (see Chapter 1). In fact there are many additive syntheses, depending on the way these partials may vary over time (no variation, linear or continuous-time variations).

2.3.1 Sinusoidal Modeling

The McAulay-Quatieri analysis [MQ86] – implemented for example in an analysis / synthesis program for MacOs called Lemur [FH96] – looks across the short-term magnitude spectra for peaks, in order to reconstruct the evolutions of the partials. These partials are pseudo-sinusoidal tracks for which frequencies and amplitudes continuously evolve slowly with time. The audio signal a in the temporal model can be calculated from the additive parameters using equations:

$$a(t) = \sum_{p=1}^{P} a_p(t) \cos(\phi_p(t)) \tag{2.5}$$

$$\frac{d\phi_p}{dt}(t) = 2\pi f_p(t) \quad \text{i.e.} \quad \phi_p(t) = \phi_p(0) + 2\pi \int_0^t f_p(u)\ du \tag{2.6}$$

where P is the (finite) number of partials and the functions f_p, a_p, and ϕ_p are the instantaneous frequency, amplitude, and phase of the p-th partial, respectively. The P pairs (f_p, a_p) are the parameters of the additive model and represent at time t points in the frequency-amplitude plane, as shown in Figure 2.13, and the evolutions of the partials in time make curves as shown

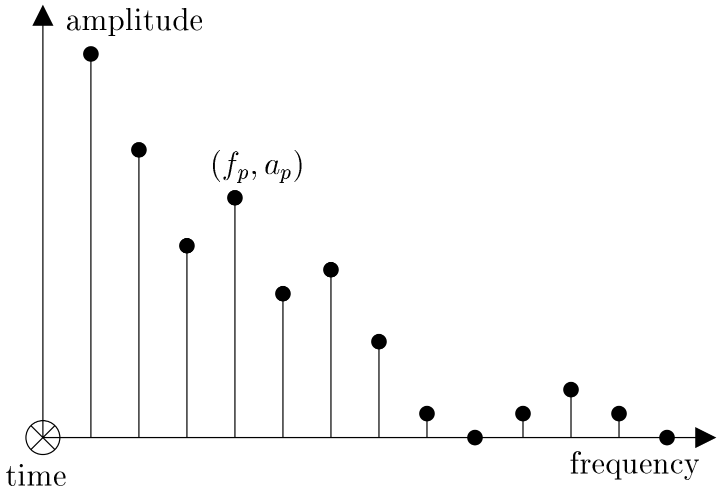


Figure 2.13: Partials of an harmonic sound.

in Figure 2.14. It is important to note that the phases of the partials are recomputed from scratch and not given by analysis. More precisely ϕ_p can be recovered using the initial phase $\phi_p(0)$ and Equation 2.6. The initial phase $\phi_p(0)$ can also be ignored during analysis and set to an arbitrary value for resynthesis. This is a consequence of psychoacoustic experiments (see Chapter 1). The choice of the $\pi/2$ value has been done in the *ReSpect* synthesis software package (see Chapter 4).

The real-time synthesis has been implemented in the *ReSpect* software tool (see Chapter 4). The difficulty is then to obtain these parameters from real, existing sounds. An accurate analysis method can be found in Chapter 3. This analysis method as well as many others have been implemented in the *InSpect* program (see Chapter 3).

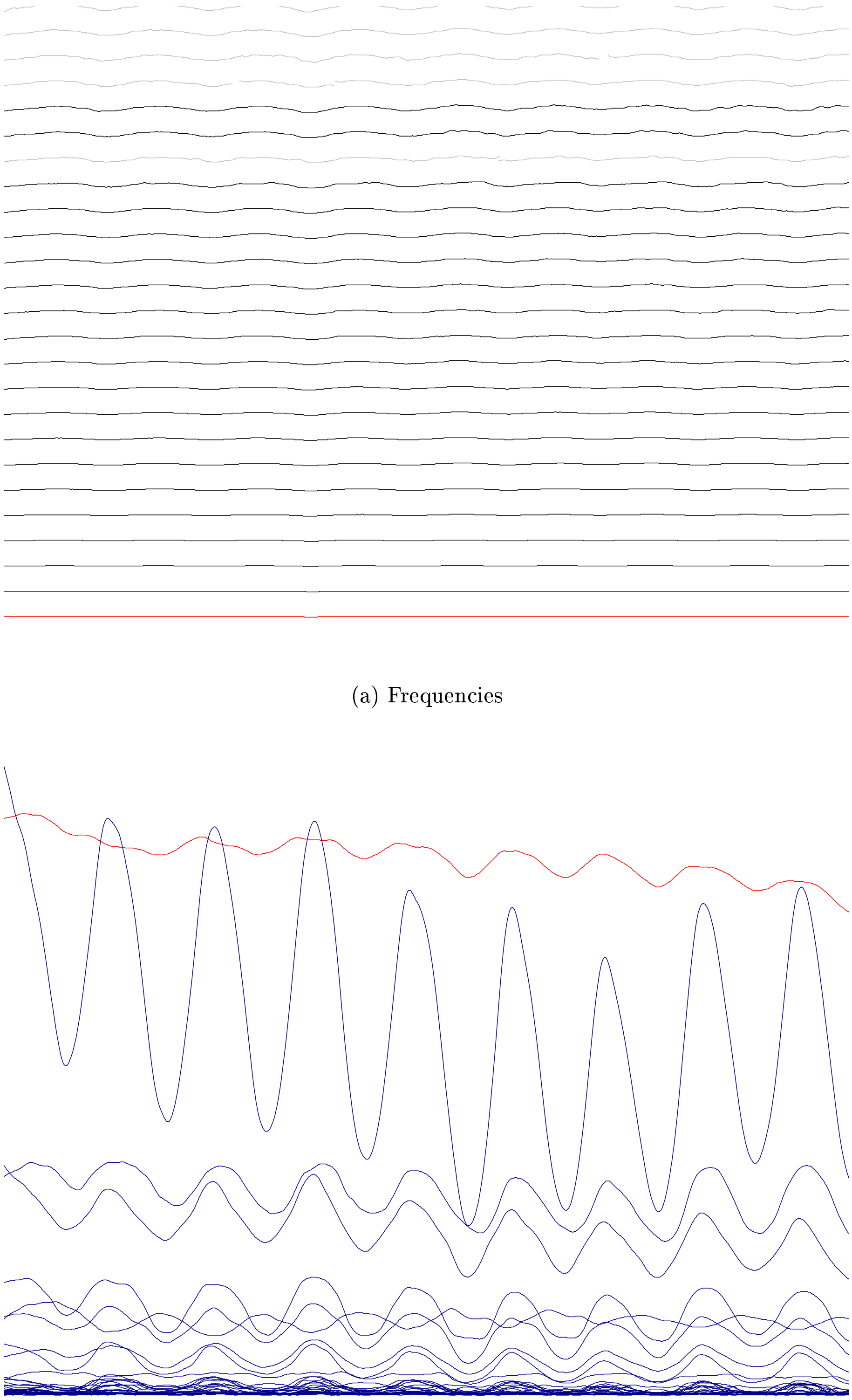
Partial Evolutions

The hypothesis that partial frequencies and amplitudes evolve slowly is indeed very important. For example, given any sound a, the solution $P = 1$, $f_1(t) = 0$, $a_1(t) = a(t)$ trivially verifies the model equations, but since a_1 is not slow time-varying such a solution is ruled out by this hypothesis. However, the definition of "slow time-varying" has to be clarified.

More formally the a_p and f_p functions are band-limited in frequency, with a small frequency much less than the value of the smallest f_p. More precisely they are band-limited to a frequency $F_{\max}$ around 20 Hz. Indeed the variations of the parameters should remain inaudible, or else modulation phenomena occur.

As a consequence, the Shannon-Nyquist sampling theorem ensures that, in theory, sampling the evolutions of the partials at 2 times $F_{\max}$ samples per second is sufficient (see Section 2.1).

Sinusoidal modeling can faithfully reproduce a wide variety of sounds, but only pseudo-periodic sounds with no noise and no transients can be represented in the sinusoidal models.



(a) Frequencies

(b) Amplitudes

Figure 2.14: The evolutions of the partials of an alto saxophone during 1 second. The frequencies (a) and amplitudes (b) are displayed as functions of time (horizontal axis).

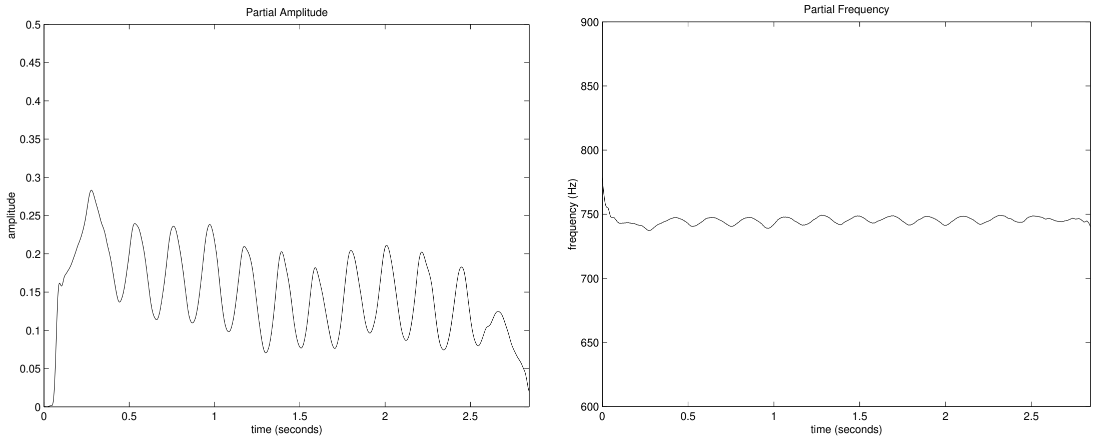


Figure 2.15: Evolutions of a partial in amplitude (left) and frequency (right).

2.3.2 Spectral Modeling Synthesis (SMS)

Spectral Modeling Synthesis (SMS) adds noise to sinusoidal modeling. Serra and Smith propose in SMS [SS90, Ser89] a spectral model based on a deterministic plus stochastic decomposition. This sound model decomposes any audio signal a into the sum of two signals:

$$a(t) = d(t) + s(t) \tag{2.7}$$

where

- $d(t)$ is the deterministic part (narrow-band component),
- $s(t)$ is the stochastic part (broad-band component).

Deterministic Part

The deterministic part consists of a sum of sinusoidal oscillators (partials) for which frequency and amplitude evolve in a slow time-varying manner. This is exactly like in sinusoidal modeling. In the very first step of the analysis, SMS searches for the parameters – amplitude, frequency, and phase – of the partials.

Stochastic Part

SMS performs the resynthesis of the deterministic part, using also the phase information, and then subtracts it from the original signal. The stochastic part is what remains after the deterministic part has been subtracted from the original signal and is generally called noise. It is in fact a random signal with statistical properties. In SMS this stochastic part is modeled as a filtered white noise.

SMS has been implemented for Windows and Linux in SMS Tools [Ser97b]. Although the very first version was open source, it is not the case anymore. SMS can reproduce pseudo-periodic sounds with filtered white noise, but still no transients.

2.3.3 Sinusoids+Transients+Noise (S+T+N)

Verma and Meng introduce in [VM98b, VM98a] the Sinusoids+Transients+Noise (S+T+N) model, which extends SMS with transients represented in the temporal model. S+T+N can reproduce pseudo-periodic sounds with filtered white noise as well as transients, provided that an accurate analysis method is able to separate the sinusoids, noise, and transients components. The main difficulty is the analysis stage. Unfortunately no free implementation seems available yet.

Sinusoids
The sinusoids are the same as in sinusoidal modeling, and as a consequence the same as in SMS. In the first step of the analysis the partials are searched for, as in SMS.

Noise
The sinusoids are then resynthesized and subtracted from the original signal, thus leading to a stochastic part as in SMS. The difference with SMS is that the second step of the analysis looks for fast variations (transients) in the stochastic part, and remove those transients from the noise.

Transients
The transients are small pieces of signal in the temporal model. They have been extracted from the noise. So the S+T+N model is an hybrid model consisting of spectral parameters together with signals expressed in the temporal domain.

2.3.4 Transformations

These additive models are extremely expressive and allow perfect filtering or time-stretching while simplifying the design of such effects too. Perfect filtering is just a matter of multiplying each amplitude a_p by the gain of the filter at the corresponding frequency f_p. Whereas time-stretching gets trivial in these models (mostly because it is possible to get rid of the phase), reverberation turns out to be impossible. In fact the reverberation of a partial is not a partial anymore. Indeed a single partial can lead to a huge – possibly infinite – number of simultaneous frequencies, so that the reverberated sound would not be in the model anymore (since the number of partials P would not be finite). Of course reverberation is a linear transformation, and does not produce frequencies that were not in the original sound. But this is not a "short-time linearity" in a sense that there can be at a certain time t frequencies that did not exist at time t in the original sound. Those frequencies are replicas of frequencies emitted before t. Figure 2.16 illustrates this phenomenon.

Anyway it is possible simply by scaling the amplitude, frequency or time axis to change, respectively, the amplitude, frequency or duration of the sound. But the frequency and the timbre are still related, as shown in Table 2.2. Cross-synthesis or pitch-shifting without shifting formants require another level of structuring for the parameters.

2.4 Structured Additive Synthesis (SAS)

The models based on additive synthesis are extremely difficult to use directly for creating and editing realistic sounds. This observation can be done in practice using software programs

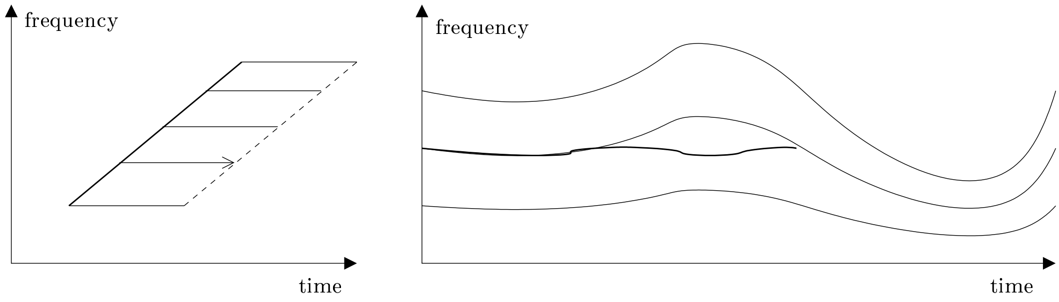


Figure 2.16: On the left, the reverberation of a single oscillator whose frequency increases linearly gives rise in theory to an infinite number of partials. On the right, after reverberation the second partial seems to "fork". In fact the reverberation process gives birth to other partials made of frequencies which are replicas of frequencies emitted before.

parameter	Amplitude	Frequency	Timbre	Time
Amplitude	×			
Frequency		×	×	
Time				×

Table 2.2: Interdependence of the musical parameters in sinusoidal modeling. When changing a certain parameter (rows), other parameters may change too (columns).

based on additive synthesis such as SoftSynth [dig]. The reason for this difficulty is the huge number of model parameters – controlling many sinusoidal oscillators – which are physically valid but only remotely related to musical parameters as perceived by a listener [Jeh97]. An interesting idea is then to group some of the parameters in order to ease their manipulation [Kle89].

Structured Additive Synthesis (SAS for short) [DCM99b, DCM99a, Mar99, Mar00d] is a spectral sound model that keeps most of the flexibility of additive synthesis while addressing these problems. It imposes constraints on the additive parameters, giving birth to structured parameters as close to perception and musical terminology as possible, thus reintroducing a perceptive and musical consistency back into the model. To validate the correspondence between the parameters of our model and the perception of music by a listener, we work in close collaboration with psycho-acousticians and composers of electro-acoustic music.

We have developed an accurate analysis method to get the model parameters from sampled sounds (see Chapter 3). It is of course possible to eliminate analysis entirely, and create new sounds directly, using the parameters of our model. This is indeed possible because there is a close correspondence between these parameters and real music perception.

2.4.1 Structured Parameters

The SAS model consists of a complete abstraction of sound according to only four physical parameters, functions closely related to perception. These parameters – amplitude, frequency, color, and warping – are inspired by the work on timbre of researchers like Risset [Ris86a, Ris86b], Wessel [Wes78, Wes79], and McAdams [McA84, MBM99], and by the vocabulary of composers of electro-acoustic music. They constitute a solid base for investigating scientific research on the notion of timbre.

We note (A, F, C, W) a sound in the SAS model. The first two parameters – amplitude A and frequency F – are one-dimensional, functions of time only, while the two others – color C and warping W – are two-dimensional, functions of both frequency and time. These two-dimensional parameters can indeed simplify the problem of controlling partials for additive synthesis, and manipulating them in a sensible way [SR99].

Amplitude

Amplitude is a function A : time $\rightarrow$ amplitude. Human beings perceive amplitude on a logarithmic scale, and the amplitude is related to the intensity which corresponds to the volume in dB. In the additive representation, the amplitude A corresponds to the sum of the amplitudes of all partials and can be calculated from the additive parameters using Equation 2.8. In order to consider the RMS (Root Mean Square) amplitude (closer to perception), Equation 2.9 must be used instead of Equation 2.8.

$$A(t) = \sum_{p=1}^{P} a_p(t) \tag{2.8}$$

$$A_{\mathrm{RMS}}(t) = \frac{1}{\sqrt{2}} \sqrt{\sum_{p=1}^{P} (a_p(t))^2} \tag{2.9}$$

When the sound is amplified, these two amplitudes have the same relative variation:

$$\begin{aligned} \forall p, \quad a_p &\rightarrow k \cdot a_p \\ A &\rightarrow k \cdot A \\ A_{\mathrm{RMS}} &\rightarrow k \cdot A_{\mathrm{RMS}} \end{aligned}$$

and since only relative values really matter to the ear, any of these two amplitudes could be used. However the RMS amplitude is closer to the perception: While A corresponds to a physical level, A_{RMS} is related to the perceptive loudness.

Calculating the volume in dB from the amplitude is easy (see Chapter 1):

$$V(A) = 20 \log_{10}\left(\frac{A}{A_{0\mathrm{dB}}}\right) \; dB \tag{2.10}$$

where $A_{0\mathrm{dB}}$ is the amplitude of reference for 0 dB. In the remainder of this section we will use the value $1/\sqrt{2}$ or 1 whether or not we are considering the RMS amplitude A_{RMS}.

RMS Amplitude

In fact, Equation 2.9 is only an approximation of the real RMS amplitude a_{RMS}, measured for a signal s in the time domain:

$$a_{\mathrm{RMS}}[s] = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (s[i])^2} \tag{2.11}$$

However, this is indeed a very good approximation. And here is why. For any partial p with frequency f, amplitude a, and initial phase ϕ_0, we have:

$$a_{\mathrm{RMS}}[a \sin(2\pi f T_s + \phi_0)] = \frac{a}{\sqrt{2}} \tag{2.12}$$

Note that the RMS amplitude does not depend on the phase. This is still true when the sound consists of a sum of partials, provided that their frequencies are different though. For two sounds, we have:

$$\begin{aligned}
a_{\mathrm{RMS}}[s_1 + s_2] &= \sqrt{\frac{1}{N}\sum_{i=1}^{N}(s_1[i] + s_2[i])^2} \\
a_{\mathrm{RMS}}[s_1 + s_2] &= \sqrt{\frac{1}{N}\sum_{i=1}^{N}((s_1[i])^2 + (s_2[i])^2 + 2s_1[i]s_2[i])} \\
a_{\mathrm{RMS}}[s_1 + s_2] &= \sqrt{\frac{1}{N}\left(\sum_{i=1}^{N}(s_1[i])^2 + \sum_{i=1}^{N}(s_2[i])^2 + 2\sum_{i=1}^{N}s_1[i]s_2[i]\right)}
\end{aligned}$$

Let us now consider sounds consisting of exactly one partial in the additive model. For two such simple sounds s_1 and s_2 with different frequencies, the cross-terms in the equation above can be neglected. More formally $\sum_{i=1}^{N} s_1[i]s_2[i] \approx 0$ because $\sum_{i=1}^{N} \sin(c_1 i)\sin(c_2 i) \approx 0$ for $c_1 \neq c_2$. This is the same argument as the one used in the Fourier decomposition. As a consequence, we have:

$$(a_{\mathrm{RMS}}[s_1 + s_2])^2 \approx (a_{\mathrm{RMS}}[s_1])^2 + (a_{\mathrm{RMS}}[s_2])^2$$

and finally, since amplitude is always positive:

$$a_{\mathrm{RMS}}[s_1 + s_2] \approx \sqrt{(a_{\mathrm{RMS}}[s_1])^2 + (a_{\mathrm{RMS}}[s_2])^2} \tag{2.13}$$

The generalization of Equation 2.13 for such n additive sounds consisting of exactly one partial leads to Equation 2.9.

Frequency

Frequency is a function F : time $\rightarrow$ frequency. The way of calculating the frequency from the additive parameters is trickier, and is explained in Chapter 3. Anyway, for harmonic sounds F coincides with the fundamental (see Figure 2.17), possibly missing or "virtual". The frequency is also perceived on a logarithmic scale, and is related to the pitch (see Chapter 1). For example, the MIDI pitch [MMA96] is a function of frequency given by equation:

$$P(F) = 69 + 12\,\log_2\left(\frac{F}{440\text{ Hz}}\right) \tag{2.14}$$

where 69 corresponds to the `A3` tone, 70 to `A#3`, etc.

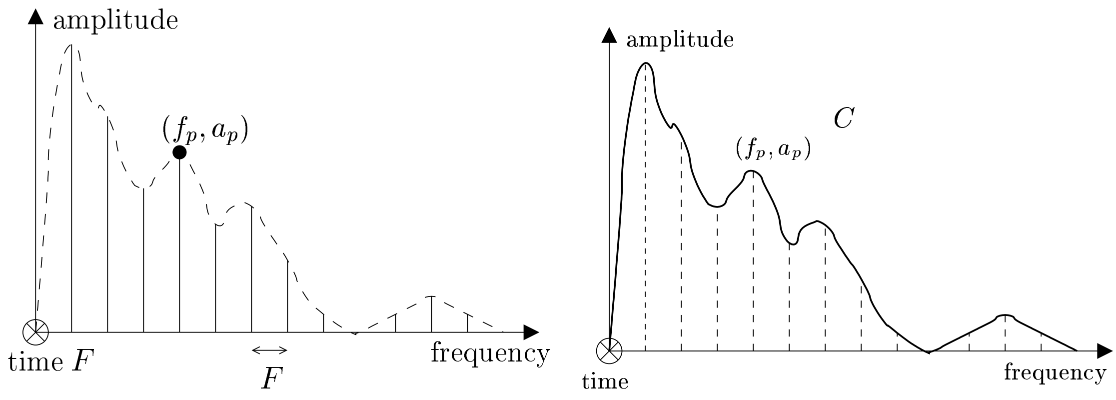


Figure 2.17: An harmonic sound at time t, with its frequency F and color C.

Color

Color is a function C : frequency $\times$ time $\to$ amplitude. Color coincides with an interpolated version of the spectral envelope [Ris86b]. The reason why we call it color is both physical and musical. There is a great analogy between audible and visible spectra. Visible colors are indeed defined by their spectral envelopes [Gla95]. This analogy is already well-known for noises (white, blue, etc.). The vocabulary of the music composers is also marked with this notion of color. They speak about the coloration induced by filters, loudspeakers or concert halls, which indeed corresponds to modifications of the spectral envelope. Color and its manipulation is amply used in contemporary popular music, though such manipulations are inherently present in the timbre of some ancient instruments like the didjeridoo, which shows very unusual color variations.

Color is a time-varying function (see Figure 2.18). Nevertheless for some instrumental sounds it may be close to a constant over time (see Figure 2.21).

Color is a continuous function. When the color is known only at the frequencies of the partials f_p, it can be completely reconstructed from these samples using uniform or irregular reconstruction methods (see Section 2.1) depending on whether the f_p are harmonically related ($f_p = p\ F$) or not.

Color is a positive function. A problem is the arbitrary choice of $C(0,t)$, since this value cannot be given by the analysis. The classic "mirroring extrapolation" technique will choose something like $C(0,t) = 2\ C(1,t) - C(2,t)$. But since $C(0,t)$ has to be positive, it may not be a good choice. Unless specified, we arbitrary use the value $C(0,t) = 0$. Schwarz and Rodet propose in [SR99] to use $C(0,t) = 1$. Moreover it is impossible that $\exists t, \forall f,\ C(f,t) = 0$. In such a case (corresponding to the silence), the other parameters would simply be undefined (corresponding to the absence of sound).

Color should be represented in a log-log scale for a better correspondence with the perception. We are envisaging interesting researches on the perception of the color, in close collaboration with psycho-acousticians. For example it turns out that it is a peculiarity of the color that it can have its own pitch, which can in some cases override the frequency of the sound and thus influence the pitch the listener actually perceives. Moreover many characteristics exist for the color itself, such as the "brightness" [Ris86b], which is related to the

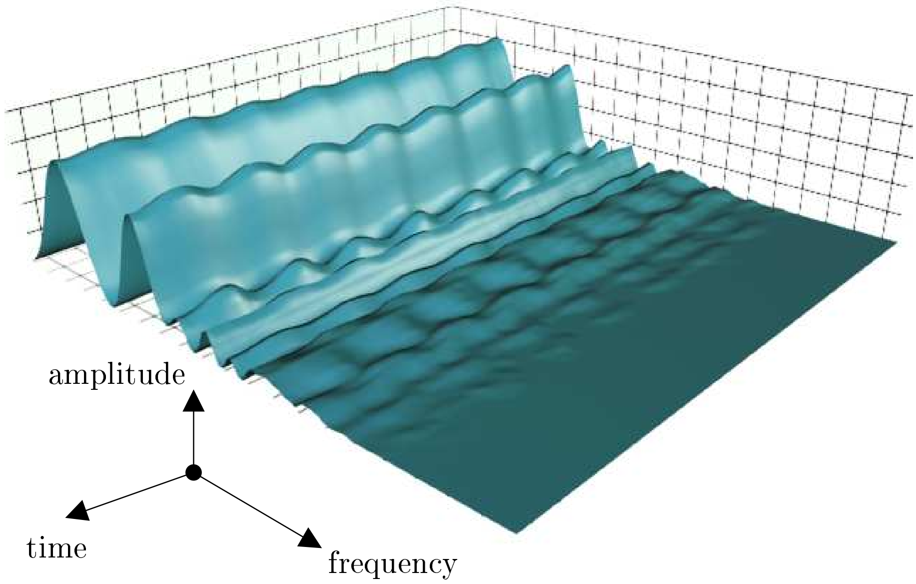


Figure 2.18: The color of an soprano saxophone with time.

spectral centroid:

$$B(t) = \frac{\sum_{p=1}^{P} a_p(t)\ f_p(t)}{A(t)} = \frac{\sum_{p=1}^{P} C(f_p(t), t)\ f_p(t)}{A(t)} \tag{2.15}$$

It might be a good idea to split the color into two colors: one for odd harmonics, the other for even ones. But we prefer to keep the analogy with the light spectrum, since it opens up new perspectives. For example, the video color is often encoded – by taking perception into account – as a Red+Green+Blue (RGB) value. It could be possible to encode in the same way the audio color by using 24 critical bands (see Chapter 1), that is with a set of 24 values instead of 3 for RGB.

For harmonic sounds the variations of the color with time constitutes the timbre itself. For non-harmonic sounds, another parameter is necessary.

Warping

Warping is a function W : frequency $\times$ time $\rightarrow$ frequency. Harmonic sounds are totally defined by the A, F, and C parameters (see Figure 2.17). But when sounds are not perfectly harmonic [MP80a, MP80b], the frequencies of the partials are not exactly multiples of the fundamental frequency F. That is why the fourth parameter – called warping after wavelet terminology [EC98] – gives the real frequency of a partial from the theoretical one it should have had if the sound had been harmonic, so that we have $f_p(t) = W(pF(t), t)$. Warping is related to inharmonicity [Rio84]. Of course for all harmonic sounds $W = \mathrm{Id}_1$, that is $\forall f, t,\ W(f, t) = f$. Some sounds have a natural warping, such as pianos, gongs or bells for example. Figure 2.19 shows such a non-harmonic sound.

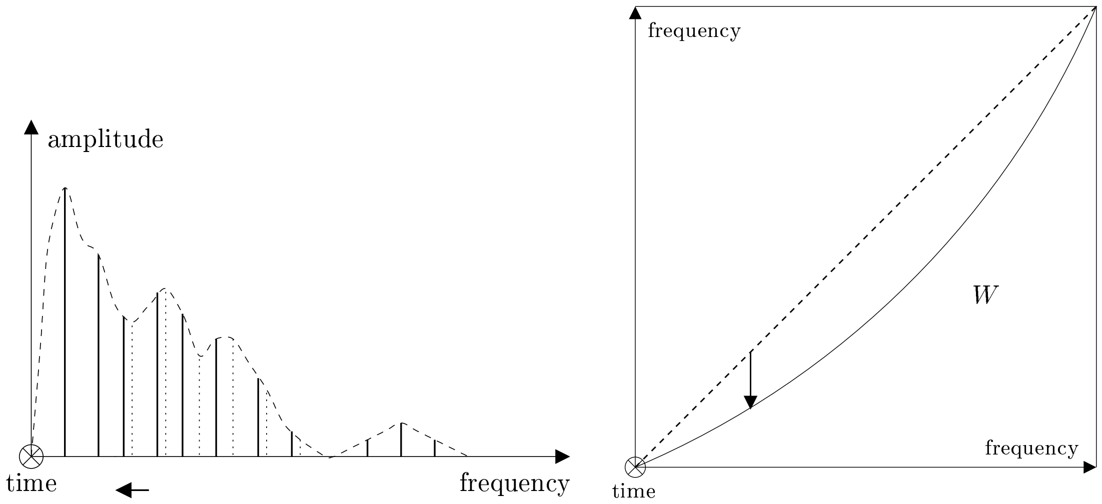


Figure 2.19: A non-harmonic sound a time t (left) with its warping envelope W (right).

Warping is a continuous function, and can be reconstructed from samples exactly like the color using the reconstruction methods explained in Section 2.1.

Warping is a positive function, and necessarily we have $W(0,t) = 0$, although this value is useless, since no partial can have a null frequency. Since partials cannot be crossing, we must have $\forall t,\ W(pF,t) > W(p'F,t)$ iff $p > p'$. So $\forall t_0,\ W(f,t_0)$ is a strictly growing function. Figure 2.20 shows some examples of warping functions. Another important point is that the “warped” frequency $W(f,t)$ should not be too far away from the original frequency f, or else the sound would be so distorted that a single SAS sound might be perceived as composed of two different sources. As a consequence there should exist a constant $K > 0$ (within a few percents) so that $\forall t,\ |W(f,t)/f - 1| < K$. Warping should also be represented in the log-log scale for a better correspondence with the perception. Since $\forall t_0,\ W(f,t_0)$ is a strictly growing function, a good idea might be to deal with its first derivative, which must then be strictly positive.

A free software implementation called *ProSpect* is being developed in order to allow the manipulation of the sound in this model (see Section 2.6).

All the SAS parameters vary slowly over time. More precisely they are band-limited to a frequency around 20 Hz. Indeed the variations of the parameters should remain inaudible, or else modulation phenomena occur and question the perceptive consistency of the parameters. The variations of the parameters reveal the inner structures of sound, and they can describe any pseudo-periodic sound provided that it is a monophonic source with no noise and no transients.

2.4.2 Structured Equations

The SAS parameters were already used – among others – to describe some characteristics of the sounds. This time these parameters are sufficient to define the sound completely. From the four structured parameters, we can calculate directly the audio signal a in the temporal model

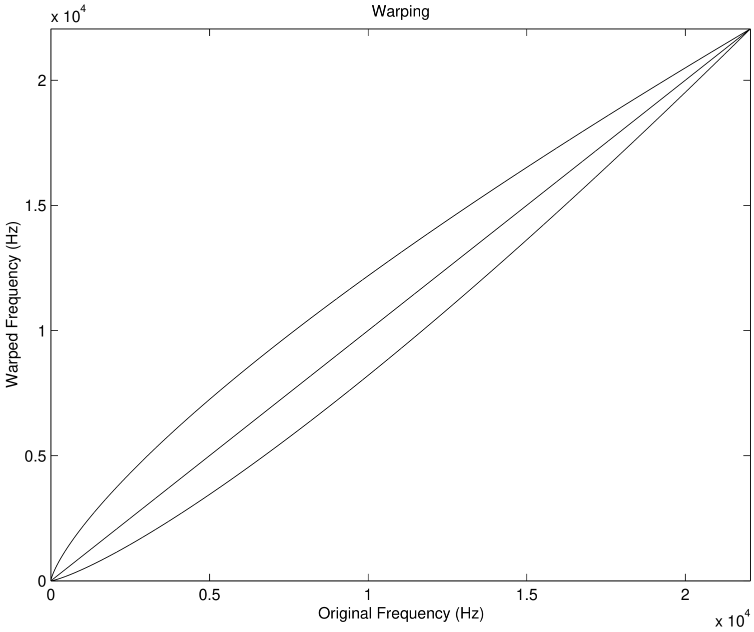


Figure 2.20: Examples of warping functions at time t.

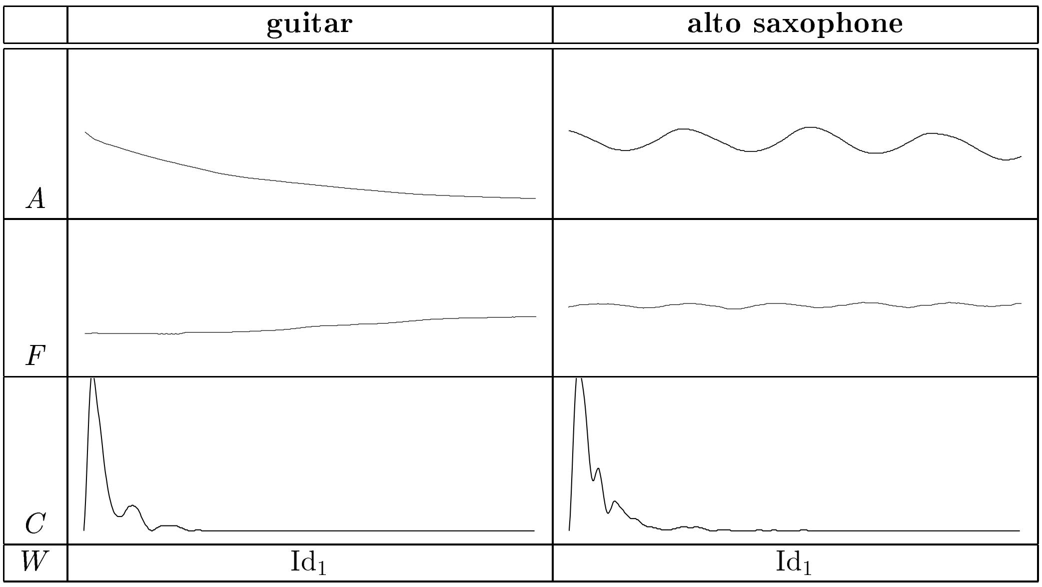


Figure 2.21: Examples of instrumental sounds in the SAS model. From top to bottom are displayed the four parameters of the SAS model: the amplitude and frequency as functions of time $A(t)$ and $F(t)$, the color as a function of frequency only $C(f)$, since it does not vary much over time for the examples considered here, and finally the warping which satisfies $W(f,t) = f$ because both sounds are harmonic.

using Equations 2.16 or 2.17, depending on whether or not the RMS amplitude is considered:

$$a(t) = A(t)\frac{\sum_{p=1}^{P} C(W(pF(t),t),t)\cos(\phi_p(t))}{\sum_{p=1}^{P} C(W(pF(t),t),t)} \tag{2.16}$$

$$a(t) = \sqrt{2}A_{\mathrm{RMS}}(t)\frac{\sum_{p=1}^{P} C(W(pF(t),t),t)\cos(\phi_p(t))}{\sqrt{\sum_{p=1}^{P} \left(C(W(pF(t),t),t)\right)^2}} \tag{2.17}$$

where $P = \max_t\{\lfloor \frac{F_{\max}}{F(t)} \rfloor\}$ ($F_{\max}$ being the highest audible frequency) and

$$\phi_p(t) = \phi_p(0) + 2\pi \int_0^t W(pF(u),u)\ du \tag{2.18}$$

These equations are the "structured" versions of Equations 2.5 and 2.6, and require approximately the same computation time. Any sound can be faithfully synthesized in real time from the model parameters using these equations (see Chapter 4).

2.4.3 Limitations and Possible Extensions

Structured Additive Synthesis (SAS) can faithfully reproduce a wide variety of sounds – as additive synthesis does – provided that they are pseudo-periodic sounds corresponding to monophonic sources. However it cannot produce noise or transients. Reverberation can also be a great disadvantage for SAS (see Section 2.5). The SAS model can be extended with symbolic structures to handle polyphonies, that is sets of monophonic sounds. It can also be extended to handle noise as well. On the other hand very short sounds like transients cannot be represented in this spectral model.

Noise

Noises could be added to the SAS model in the same way as in SMS, since many noises can be modeled as filtered (colored) white noises at certain amplitudes. The amplitude and color parameters exist also for noises and are sufficient to define many of them. White noise has a white color ($C = 1$), and every noise named after an analogy with a light spectrum matches this correspondence of terminology. As a consequence we can easily extend the SAS model to include noises. However amplitude and color may not be sufficient to define any kind of noise, as stated by Hanna [Han00].

Symbolic Structures

We are developing a sound synthesis language based on SAS, in close collaboration with composers of electro-acoustic music. In order to use SAS for the whole compositional process, a hierarchical model with symbolic structures must be designed on the top of the sub-symbolic structures of SAS and incorporated in the language. For that purpose we use hierarchical structures similar to the hierarchic temporal organization of musical structures proposed by Balaban [BS93]. More precisely, the elementary musical structure is

$$\{A, F, C, W\}$$

then the sequence of n musical structures is

$$[|(S_i = \{A_i, F_i, C_i, W_i\})_{i=1,n} \longmapsto \{A, F, C, W\}]$$

parameter	Amplitude	Frequency	Timbre	Time
Amplitude	×			
Frequency		×		
Timbre			×	
Time				×

Table 2.3: Independence of the musical parameters in the SAS model. It is possible to change one parameter while leaving the others unchanged.

while the superposition of n musical structures is

$$[-(S_i = \{A_i, F_i, C_i, W_i\})_{i=1,n} \longmapsto \{A, F, C, W\}]$$

Chords are represented as superpositions of elementary sounds S_i. By considering the (F_i, A_i) pairs, we have "pseudo-partials" and we can again define color and warping parameters for the chords, and the warping is this time related to the inharmonicity of the chords.

This further level of structuring – allowing for instance the manipulation of sets of SAS sounds – makes more things possible, like representing polyphonic sounds or performing echoing or mixing.

One advantage in the SAS model is the independence of the musical parameters. The amplitude, frequency, and timbre can indeed be changed independently. It is possible to stretch the sound in time without changing its pitch, to shift the pitch while preserving the timbre, and so on. Table 2.3 illustrates this independence. This property amply facilitates the design of digital audio effects in the SAS model.

2.5 Musical Sound Transformations

In the SAS model many effects become accessible not only to engineers, but also to musicians and composers. Among these are the transformation of a single sound, such as filtering or time-stretching, as well as the creation of hybrid sounds by the combination of several sounds, such as cross-synthesis or morphing. These effects turn out to be straightforward in the SAS model and can be designed in a very intuitive way. This is done at the expense of restrictions not only on the kind of sounds that can be represented in the SAS model, but also on the kind of effects that can be performed.

2.5.1 Transforming Sounds

Given an SAS sound $S = (A, F, C, W)$, the simplest sound transformations can be expressed as a simple multiplication on one of its SAS parameters. In the remainder of this section we use the standard notation for the product between functions, that is for one-dimensional parameters $(kP)(t) = k(t)\ P(t)$ while for two-dimensional ones $(kP)(f, t) = k(f, t)\ P(f, t)$.

Amplitude, Amplification, and Tremolo

Changing the volume of S is trivial: Given an amplification factor k, consider the sound (kA, F, C, W). If k is a constant, this is a simple amplification. If k is a sinusoid with a

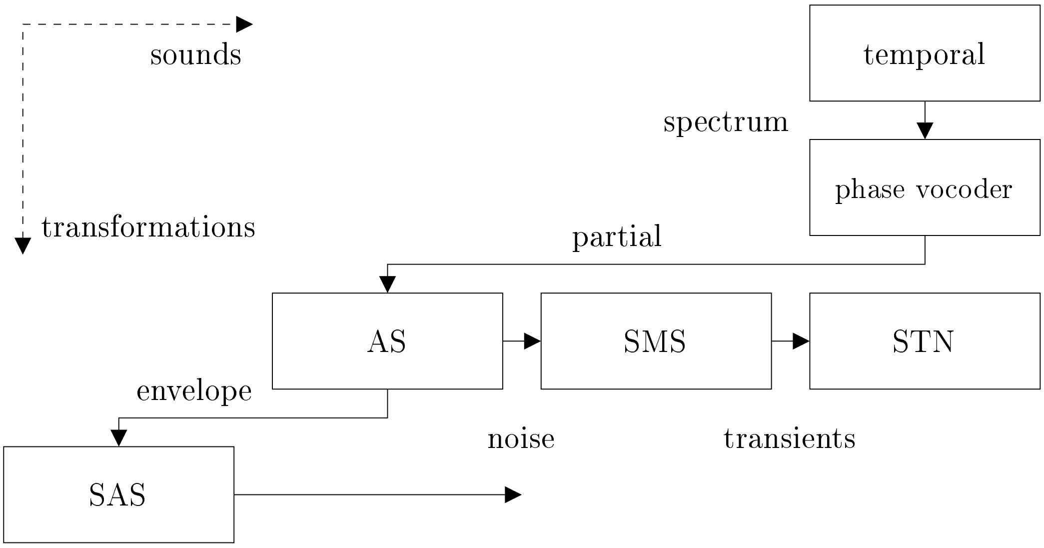


Figure 2.22: Expressiveness of several sound models. The temporal, phase vocoder, additive synthesis (AS), Spectral Modeling Synthesis (SMS), Sinusoids+Transients+Noise (STN), and Structured Additive Synthesis (SAS) models are compared. The horizontal axis represents the variety of sounds that can be represented (the models at the right being the most general), whereas the vertical axis represents the ease for performing musical transformations (the models at the bottom being the best-suited for these transformations).

frequency around 8 Hz, the musical effect obtained is a tremolo with this frequency. If the variations of k are slow and monotonous (mathematically speaking), this is either a fade-in or a fade-out.

Frequency, Transposition, and Vibrato

Pitch-shifting is quite as much easy: Given a transposition factor k, consider (A, kF, C, W). The base-2 logarithm of k is the amount of pitch-shifting expressed in octaves. Figure 2.23 shows a sound and the same sound one octave lower ($k = 0.5$). If k is a constant, this is a simple transposition. If k is a sinusoid with a frequency around 8 Hz, the musical effect obtained is a vibrato with this frequency. If the variations of k are slow and monotonous (again mathematically speaking), this a glissando or a portamento.

Color and Filtering

When k is the spectral response (color) of a filter, filtering (coloring) S with this filter can be done ideally like this: $S' = (A, F, kC, W)$. This is a perfect filtering, without artifacts. In fact the design of a filter in the SAS model is mainly the specification of the color parameter for the filter, which can of course evolve in time.

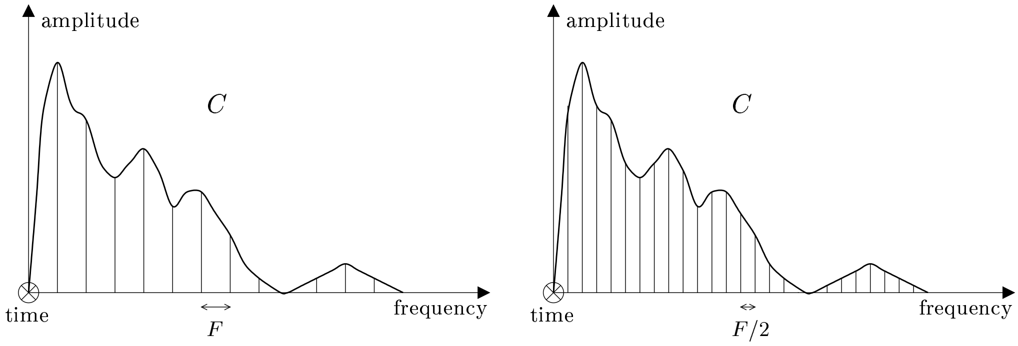


Figure 2.23: Two sounds with the same color but different frequencies.

Warping and Harmonization

Warping S sounds a bit stranger, since we are not really used to this parameter in music. However we are carrying out promising experiments in close collaboration with composers of electro-acoustic music. It has already been used in several pieces to draw a link between harmonic and non-harmonic sounds (see Chapter 5). Warping is related to inharmonicity, and one can change it easily like this: $S' = (A, F, C, kW)$, where k is a "warping envelope". One can also perform a new kind of cross-synthesis by replacing the warping of one sound by the one of another sound.

Time and Time-Stretching

Since all the SAS parameters are functions of time, time-stretching is only a matter of scale on the time axis. For example the sound $(A(kt), F(kt), C(kt), W(kt))$ is k times shorter than S. Of course k does not have to be a constant. Time-varying stretching factors can produce very impressive effects.

2.5.2 Combining Sounds (Hybrid Sounds)

One of the advantages of the SAS model is its aptitude for creating hybrid sounds from the combination of several sounds.

Cross-Synthesis

One can perform many kinds of cross-syntheses only by interchanging parameters among different sounds. Figure 2.24 illustrates a cross-synthesis on the color parameter between S_1 and S_2.

Morphing

Of course one can also blend the parameters of different sounds. Let us realize a morphing in the SAS model. Assuming that we perceive all the parameters logarithmically, consider the blending operator:

$$\forall \alpha \in [0,1], \quad \text{blend}(P_1, P_2, \alpha) = P_1^{(1-\alpha)} P_2^{\alpha}$$

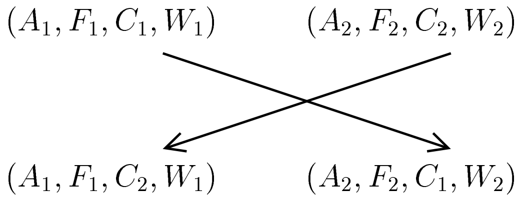


Figure 2.24: Cross-synthesis by color swapping.

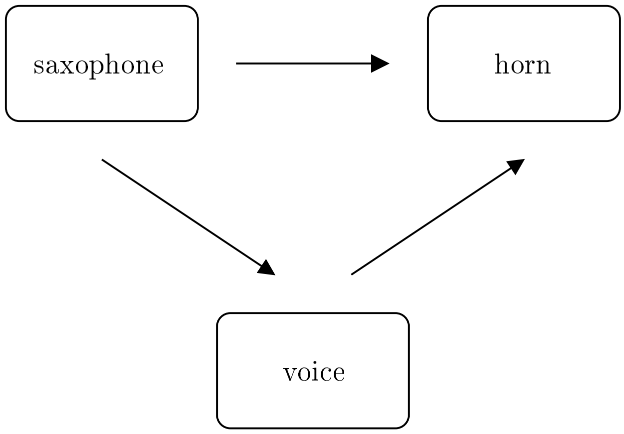


Figure 2.25: Some morphing examples (saxophone to horn, saxophone to voice, voice to horn).

By blending each parameters and making α vary over time from 0 to 1, one hears a morphing from the first sound to the second. Figures 2.25 and 2.26 illustrate this. But the *blend* operator is not well-suited for voice, since it does not really care about formants. In order to perform "formant-morphing", the blending operator has to be changed for the color parameter. In the example of Figure 2.27, the morphing should be able to go from a vowel to the other while mimicking a real singer. Sequences of morphing among vowels become then possible, like $a \to e \to i \to o \to u$. Then, by also controlling the amplitude and the frequency parameters, it is possible to realize a singing voice like the one in Figure 2.28 (see also Chapter 5).

2.5.3 About Impossible Effects

Whereas the design and control of most audio effects become considerably simplified in the SAS model [Mar99, Mar00d], some effects turn out to be impossible to achieve in this model.

While structuring the sound representation facilitates the design and control of musical sound effects, it also imposes limitations not only on the sounds that can be represented but also on the effects that can be performed on these sounds.

Reverberation, as in additive synthesis, cannot be implemented. But since SAS models (monophonic) sound sources – and not sounds reaching our ears – reverberation has no meaning in the SAS model. Echoes cannot be done either, since echoing one (monophonic) sound may lead to a polyphonic sound that cannot be represented as one single sound in the SAS model. Even the mixing of two sounds in the SAS model is not a sound in the model, since the mixing of several monophonic sources is not (in general) a monophonic source (see Figure 2.29). To

Figure 2.26: The color of sounds while morphing from a saxophone to a horn. From top to bottom are displayed the colors of the source (saxophone), the intermediate hybrid sound (half-saxophone, half-horn), and the destination (horn). The axis are the same as in Figure 2.18.

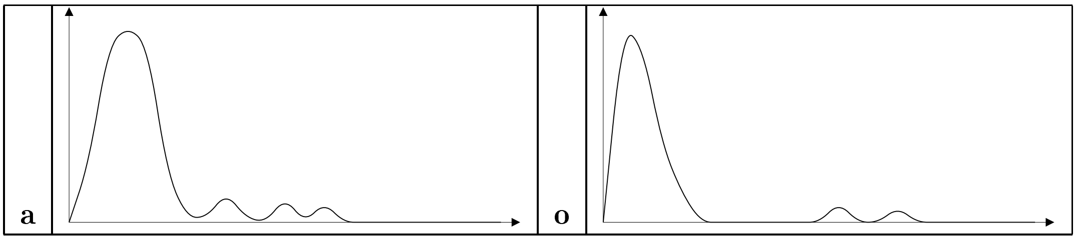


Figure 2.27: Color examples for two vowels (only the function of frequency $C(f)$ is represented since C is nearly constant in time here).

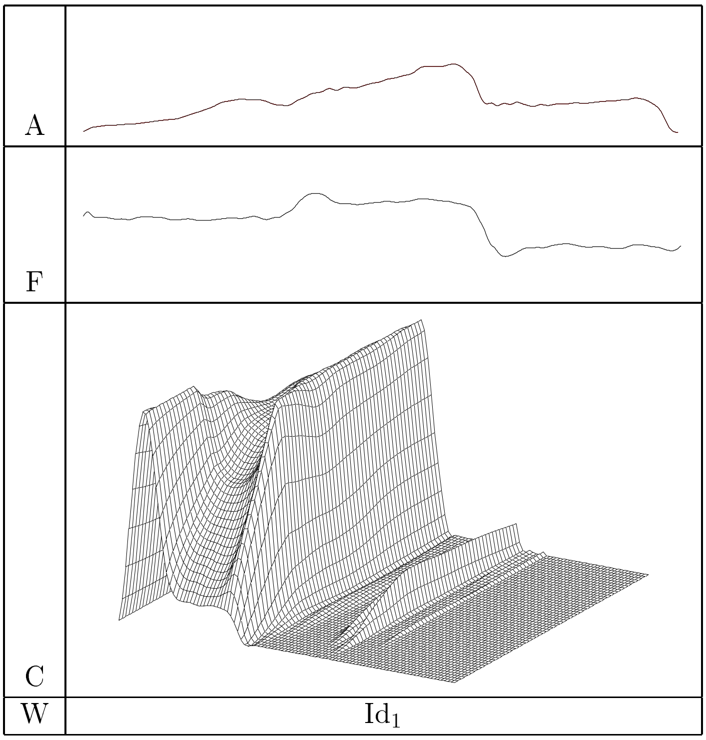


Figure 2.28: Singing voice in the SAS model.

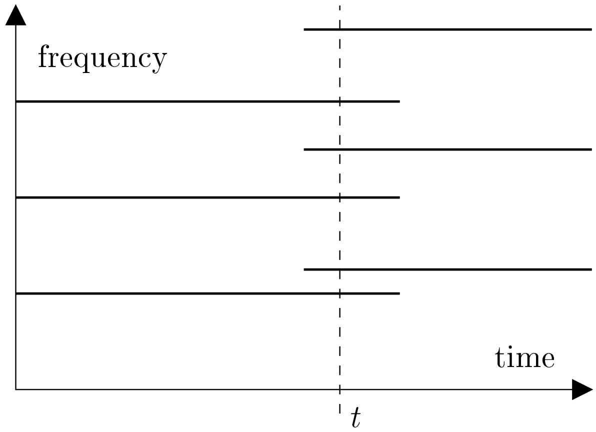


Figure 2.29: The mixing of two monophonic sources is not (in general) a monophonic source, that is why mixing is impossible in the SAS model.

manipulate polyphonic sounds – that is sets of monophonic sounds – a symbolic structuring must be added on the top of SAS, as mentioned above.

2.6 *ProSpect* Software Package

ProSpect ("Prospect Spectrum") [Mar00e] is a free software platform that allows the manipulation (analysis, transformation, synthesis) of sounds in various sound models, mainly spectral ones. It is a research tool also useful for creation (see Chapter 5). This hardware-independent architecture is developed for the Linux operating system, although a Windows 95/98/NT version exists too. It is freely distributed [Mar00b] under the terms of the GNU General Public License (GPL) [FSF91]. The development version of this tool has been tested by composers of electro-acoustic music who were enthusiastic [MB00]. This software system should open new horizons for both researchers and composers. However the current development version needs a simplification of its general architecture and still lacks a complete documentation...

ProSpect offers the possibility to act on the sounds directly via the instructions of a functional programming language. It combines the advantages of the C programming language with the power of functional languages (Scheme or Lisp). It allows to literally program the musical sound, thus facilitating the musical composition process greatly. It provides traditional operations such as filtering, time-stretching (without limits on the stretching factor), cross-synthesis, transposition while preserving formants, timbre morphing and many more. Since these manipulations are made on a spectral representation which has fewer data points than the signal itself, our operations are quite efficient. Although most operations on sound are very fast thanks to the power of modern computers, the manipulation of sounds in real time is not the goal of *ProSpect*.

Although *ProSpect* can be used alone, it is not a "final" software program such as SMS [Ser97b], but rather an open development environment in the spirit of the ATS library [Pam99] and can be used as a starting point for other software programs. *ProSpect* is at the root of software tools covering various fields from sound analysis (*InSpect*, see Chapter 3) to assistance for music composition (*BOXES*, see Chapter 5). Its opening and its extensibility allow to incessantly add new functionalities. We hope that in the future more interested people will be involved in its development and will use it as a starting point for many other software programs in the field of computer music.

2.6.1 Software Architecture

The software architecture of *ProSpect* consists of a set of basic functionalities dynamically extended by plug-in modules.

Base

This architecture results from the extension of an all-purpose functional language by a set of sound-specific data types and primitives. More precisely, the base of *ProSpect* consists of a Scheme interpreter extended by a library containing new data types and primitives mostly written in C programming language for performance reasons. There is a similar structure in ATS [Pam99], except that in our case the functional language is not Common Lisp but Scheme.

Functional Language

In the current version, *STk* [Gal99, Gal95] is used as the basic Scheme environment. *STk* (Scheme + *Tk*) is a free Scheme interpreter with native support for the *Tk* graphical toolkit. However many other Scheme interpreters exist and for example the use of *Guile* in the forthcoming versions is possible. *ProSpect* inherits the basis of λ-calculus and symbolic programming from the functional language. There is no a priori distinction either between code and data regarding the computer programming aspects or between instrument and score concerning music composition. As a consequence this artificial boundary between instrument and score present for example in CSound [Ver86, Ver92] and MPEG-4 [VGS98] does not exist anymore. Nevertheless *ProSpect* is not completely written in Scheme. Unlike Common Lisp Music [LLP99] – entirely written in Common Lisp – *ProSpect* takes advantage of extensions that are written in C, which is a lower-level language but also a more efficient one in terms of computation speed.

Graphical User Interface

ProSpect is based on *STk* (Scheme + *Tk*), and thus inherits from it not only the Scheme interpreter but also the *Tk* graphical toolkit, which offers a wide variety of graphical objects (widgets) and primitives. *ProSpect* also provides the user with new functions to facilitate the management of menus or dialog boxes for example.

Digital Signal Processing

The initial possibilities of the functional language are extended by a library of new data types and primitives implemented in C programming language. Thus, one can easily manipulate for example complex numbers, vectors or matrices in a very efficient way. Among the available extensions are new mathematical functions, linear algebra, as well as primitives for encoding and compressing data flows. *ProSpect* also contains functions that come from psychoacoustics, even if digital signal processing remains the main goal of the extensions. Of course it contains many basic tools issued from signal theory, such as analysis windows (Hann, Hamming, Blackman, Kaiser, etc.), algorithms for filtering or resampling, as well as the Fourier transform for example. By the way the Fast Fourier Transform used in *ProSpect* is the so-called "Fastest Fourier Transform in the West" (FFTW) [FJ98] developed at the MIT, which is freely distributed under the terms of the LGPL too.

Extensions

The software base can easily be extended using plug-in modules written in Scheme or in C programming languages for example. Using extensions it is possible to dynamically add sound models, analysis or synthesis methods, as well as file formats. The software programs derived from *ProSpect* can define their own extensions, which can in turn be reused in new programs. Defining such an extension is easy. The following example defines a new file format for spectral sounds:

```
(prospect-link "msc")  ; dynamic linking of msc.so

(plugin-insert-spectral-format
 '("Spectral Model (Compressed)"
   (".msc")
   spectral-test-msc
   spectral-load-msc
   spectral-save-msc))

(prospect-provide "msc")
```

This format is called "Spectral Model (Compressed)" and has a default filename extension `msc`. The `test` function decides whether of not a given file is in this format, the `load` function loads a file within this format, and the `save` function saves a spectral sound in a file using this format. These three functions are defined in the `msc.so` file, which is dynamically linked at the beginning of the example and contains the object code resulting from the compilation of a C program. *ProSpect* provides the user with all the primitives necessary for the interface between the C and Scheme languages. Here is the glue code between C and Scheme as well as the interface of the C part of the extension module above:

```
#include "plugin_scm.h"
#include "msc.h"

PLUGIN_SPECTRAL_FORMAT_DEFINE (msc, test_msc, load_msc, save_msc);
```

where file `msc.h` is:

```
#ifndef __SPECTRAL_FORMATS_MSC_H__
#define __SPECTRAL_FORMATS_MSC_H__

#include <prospect.h>
#include "spectral.h"

extern bool     test_msc (char *name);
extern spectral load_msc (char *name, ...);
extern bool     save_msc (spectral s, char *name, ...);
```

```
#endif /* !__SPECTRAL_FORMATS_MSC_H__ */
```

2.6.2 Sound Models

Sound models are extensions of *ProSpect* too. Currently there are mainly three models available:

- The temporal model (`temporal`), where each sound is represented by the acoustic pressure level at a certain point in space, as a function of time;
- The additive synthesis model (`spectral`), where sounds consist of a set of partials, sinusoidal oscillators for which amplitudes and frequencies evolve slowly in time;
- The structured additive synthesis model (`sas`), structuring the parameters of the classic additive synthesis to give birth to only four new parameters closer to the perception.

It is possible to define a sound from a file (`load` function) and inversely to create such a file from a sound defined in memory (`save` function). With these simple functions performing the conversion among different file formats but within the same sound model is easy. The following example converts a sound in the temporal model from the WAV format to the AIFF format:

```
(temporal-save (temporal-load "sound.wav") "sound.aiff")
```

The main sound formats are supported, such as AIFF or WAV for the temporal model. In fact the list of the file formats supported by *ProSpect* gets longer day after day since it is very easy to add a new format, which is inserted in the general architecture as a plug-in module. *ProSpect* supports many file formats for spectral sounds, compressed or not. The support for the Sound Description Interchange Format (SDIF) file format [WCF$^+$99, IRC00, CNM00] is under development.

One can also get the sampling rate of the sound, its size in samples as well as its duration in seconds. It is possible to normalize the amplitude of a sound or to change its sampling rate for example. One can also play the sound, even spectral ones. The underlying real-time synthesis is performed by our *ReSpect* synthesis module, via the SAS library for the SAS model (see Chapter 4).

Even if there are some common functionalities among the different sound models, the set of manipulations that can be performed on a given sound depends on the model in which this sound is expressed.

Temporal Model

In the temporal model the sound is represented as the acoustic pressure level at a certain location in space. Every band-limited sound can be represented as a stream of samples resulting from the sampling of this pressure level. Recording and reproducing such a sound can be done quite easily using, respectively, analog-to-digital and digital-to-analog converters, but the manipulation of the sound is difficult and often not intuitive. The following example records 10 seconds of sound and normalizes its amplitude before playing it back:

```
(define sound (temporal-record 10))
(temporal-normalize! sound)
(temporal-play sound)
```

The user can get the number of samples in the sound (`size`) as well as the duration of the sound in seconds (`time`), but it is impossible to modify them. On the other hand it is possible to get and change the sampling rate (`rate`) of any temporal sound. In order to go from the 48000-Hz DAT (Digital Audio Tape) format to the 44100-Hz audio CD format, while inevitably loosing the frequency contents of the original sound above $44100/2 = 22050$ Hz, one can type:

```
(define s (temporal-resample (temporal-load "sound.wav") temporal-rate-cd))
```

It is also possible to construct new sounds using superpositions (`mix`) and sequences (`seq`) of existing sounds, provided that they all have the same sampling rate. The following example constructs the `s4` sound consisting of the superposition of the `s1` and `s2` sounds followed in sequence by the `s3` sound:

```
(define s1 (temporal-load "sound1.aiff"))
(define s2 (temporal-load "sound2.aiff"))
(define s3 (temporal-load "sound3.aiff"))
(define s4 (temporal-seq (temporal-mix s1 s2) s3))
```

In fact the number of manipulations allowed in the temporal model is very limited. On the one hand the amplitude of the sound can be easily modified. On the other hand its frequency, duration, and timbre are interdependent and it is extremely difficult to change any of these parameters without changing the others.

Spectral models do allow to separate these parameters, since they are dealing with the frequency contents of the sound at certain times. They parameterize sound at the (human) receptor, possibly taking perception into account.

Additive Synthesis (AS) Model

The McAulay-Quatieri analysis [MQ86], implemented in Lemur [FH96], extracts partials from the sounds. These partials are sinusoidal oscillators for which the amplitudes and frequencies evolve slowly with time. These evolutions are sampled and it is possible, as in the temporal model, to obtain and modify their sampling rate (`rate`). It is also possible to get the number of partials, the maximum number of simultaneous partials (`polyphony`), as well as the frequency and amplitude of each partial. One can also perform time-stretching or perfect filtering, that is filtering without artifacts. It is possible to amplify or transpose a sound, but this transposition does not take the formants of the sound into account.

Although the primitives on spectral sounds are too numerous to be enumerated in details, many of them are illustrated in the remainder of this chapter.

This additive model can reproduce a wide variety of sounds, provided that they are without noise or transients though. However sound models based on additive synthesis are often difficult to use for creating or editing sounds. The reason for this difficulty is the huge number of model parameters that are only remotely related to musical ones as perceived by a listener.

Structured Additive Synthesis (SAS) Model

In the Structured Additive Synthesis (SAS) model (see Section 2.4) every monophonic sound can be described in terms of amplitude, frequency, color, and warping. All these parameters vary slowly over time. This model favors the unification of the representations of sound and music at a sub-symbolic level, since the definition of a sound or a musical operation can be done in the same way in this model (see Chapter 5).

A sound S in the SAS model is: (A, F, C, W), where the two first parameters – amplitude A and frequency F – are one-dimensional, functions of time only, while the two others – color C and warping W – are two-dimensional, functions of both frequency and time. All these parameters are functions that vary slowly over time:

Amplitude	A : time $\rightarrow$ amplitude
Frequency	F : time $\rightarrow$ frequency
Color	C : frequency $\times$ time $\rightarrow$ amplitude
Warping	W : frequency $\times$ time $\rightarrow$ frequency

In the current implementation of the SAS model, four functions get the parameter values at a given time (`sas-a-get`, `sas-f-get`, `sas-c-get`, and `sas-w-get`), and four other functions allow to change these values (`sas-a-set!`, `sas-f-set!`, `sas-c-set!`, and `sas-w-set!`). It is then possible to perform all the other necessary manipulations on these parameters directly in Scheme, as explained later.

In fact two other sound models also exist in *ProSpect*. The "harmonic" model places a strong constraint of proportionality among the frequencies of the partials. This is indeed a restricted version of the SAS model, the notion of warping being absent. However the amplitude, frequency, and color of sound can still be manipulated in this model. The second sound model is the one of the phase vocoder [Moo78c, Ser97a], that uses the short-time Fourier transform in order to produce a series of short-term spectra taken at successive times in the temporal sound. This model is not directly implemented in *ProSpect*, but is part of the *InSpect* sound analysis software tool described in Chapter 3.

2.6.3 Examples of Sound Transformations

It is possible to modify the duration, amplitude, frequency, and timbre of the sounds.

Time

Changing the duration of a spectral sound is very easy. The following example calls the `stretch-time` function that allows here to stretch the `s0` sound in time, from the starting time 0 to the end of the stretched sound, using a constant stretching factor. A stretching factor of 1 would have left the original sound unmodified.

```
; original
(define s0 (spectral-load "cello.msc"))
; 2 times slower
(define s1
 (spectral-stretch-time s0 0 (* (spectral-time s0) 2) (lambda (t) 0.5)))
; 2 times faster
(define s2
 (spectral-stretch-time s0 0 (* (spectral-time s0) 0.5) (lambda (t) 2)))
```

However the function of time used as a parameter is not necessarily constant. It can be considered as a signal, with variations much slower than the signal of the resulting sound. This mechanism allows to realize a multi-scale composition [Vag98] (see Chapter 5).

Amplitude and Frequency

Amplifying or transposing a spectral sound is just as easy. The parameters of the amplification or transposition functions are the original sound (`s0` here) and, respectively, an amplification or a transposition factor that may vary over time.

```
; fade-in
(define (fade-in s) (lambda (t) (/ t (spectral-time s))))
(define s3 (spectral-amplify s0 (fade-in s0)))

; higher octave
(define s4 (spectral-transpose s0 (lambda (t) 2)))
```

Color and Warping

Perfect filtering of a spectral sound is easy too. It is also possible to change the harmonicity of a sound thanks to the warping. The following examples illustrate these possibilities:

```
(define (filter f a d t) (* a (- 1 (/ f 6000))))
(define s5 (warp-amplitude s0 filter))

(define (warper f a d t) (pow f 1.04))
(define s6 (warp-frequency s0 warper))
```

The above filter (`filter`) is in fact a function returning the gain of the filter as a function of frequency `f`, but also of amplitude `a`, duration of the sound `d`, and current time `t`. This complex way of defining filters allows to design extremely interesting filters. But the simplest way to define a filter is to use the fact that in the SAS model a filter is simply defined by its color as a function of time:

```
(define (sas-filter! sound filter)
 (do-loop 0 (sas-size sound) 1  ; sound from the beginning to the end
  (lambda (i)
   (sas-c-set! sound i
    (sas-envelope-multiply (sas-c-get sound i) (sas-c-get filter i))))))

(define (sas-example)
  (let* ((source (sas-load "source.sas"))
         (filter (sas-load "filter.sas")))
   (sas-filter! source filter)
   (sas-save source "result.sas")))
```

2.6.4 Examples of Sound Hybridizations

One of the advantages of the SAS model is its aptitude for creating hybrid sounds.

Cross-Synthesis by Color Swapping

One can perform many kinds of cross-syntheses only by interchanging parameters among different sounds. Figure 2.24 illustrates a cross-synthesis on the color parameter between S_1 and S_2. Programming such a transformation within *ProSpect* is trivial:

```
(define (sas-cross! s1 s2)
 (do-loop 0 (max (sas-size s1) (sas-size s2)) 1
  (lambda (i)
   (let* ((c1 (sas-c-get s1 i))
          (c2 (sas-c-get s2 i)))
    (sas-c-set! s1 i c2)
    (sas-c-set! s2 i c1)))))
```

Simple Morphing

Of course one can also blend the parameters of different sounds. The following program realizes a simple morphing from one sound to a second one, without taking formants into account:

```
(define (morpher v1 v2 alpha)
  (* (pow v1 (- 1 alpha)) (pow v2 alpha)))

(define (sas-envelope-blend e1 e2 morpher alpha)
 (if (= (vector-length e1) (vector-length e2))
  (let ((e (make-vector (vector-length e1))))
   (do-loop 0 (vector-length e) 1
    (lambda (i)
     (vector-set! e i (morpher (vector-ref e1 i) (vector-ref e2 i) alpha))))
   e)))

(define (sas-morphing s1 s2 morpher)
 (let* ((n (max (sas-size s1) (sas-size s2)))
        (s (sas-make n)))
  (do-loop 0 n 1
   (lambda (i)
    (let* ((alpha (/ i (- n 1)))
           (a1 (sas-a-get s1 i))
           (f1 (sas-f-get s1 i))
           (c1 (sas-c-get s1 i))
           (w1 (sas-w-get s1 i))
           (a2 (sas-a-get s2 i))
           (f2 (sas-f-get s2 i))
           (c2 (sas-c-get s2 i))
           (w2 (sas-w-get s2 i))
           (a (morpher a1 a2 alpha))
           (f (morpher f1 f2 alpha))
           (c (sas-envelope-blend c1 c2 morpher alpha))
           (w (sas-envelope-blend w1 w2 morpher alpha)))
```

```
      (sas-a-set! s i a)
      (sas-f-set! s i f)
      (sas-c-set! s i c)
      (sas-w-set! s i w))))
   s))

(define (sas-example)
 (let* ((s1 (sas-load "source1.sas"))
        (s2 (sas-load "source2.sas"))
        (s (sas-morphing s1 s2 morpher)))
   (sas-save s "target.sas")))
```

This algorithm was used to generate the sounds illustrated in Figure 2.26.

Chapter 3

Sound Analysis

Digital sound synthesis and manipulation require a good model of sound (see Chapter 2). Spectral models provide general representations in which such operations can be performed in a very natural and musically expressive way. In order to faithfully imitate or transform existing sounds, such models require an analysis method to extract spectral parameters from sounds which were usually recorded in the temporal model, that is audio signal amplitude as a function of time. The accuracy of the analysis method is extremely important since the perceived quality of the resulting spectral sounds depends mainly on it. The main interest of an accurate analysis method, providing precise parameters for the spectral models, is to allow ever deeper musical transformations on sound by minimizing deformations due to analysis artifacts.

Designing an accurate analysis method is a difficult problem. This chapter presents some analysis methods for extracting spectral parameters from sounds originally in the temporal model. For those analysis methods to be efficient, the spectral model considered has to be restricted.

The remainder of this chapter will focus on sounds for sinusoidal modeling. In practice, this restriction means that the considered sounds should have a low noise level, which is true for many clear natural sounds, perhaps after their attack phase. Thus, this chapter only concerns an analysis method for extracting parameters for the deterministic part of the sounds. The deterministic part is the sum of sinusoidal oscillators whose frequencies and amplitudes evolve in a slow time-varying manner. Such a slow time-varying oscillator is commonly called a *partial*. More formally:

$$a(t) = \sum_{p=1}^{P} \mathrm{osc}(f_p(t), a_p(t), \phi_p(0)) \tag{3.1}$$

where P is the number of partials and

$$\mathrm{osc}(f_p(t), a_p(t), \phi_p(0)) = a_p(t)\cos(\phi_p(t)) \tag{3.2}$$

with

$$\frac{d\phi_p}{dt}(t) = 2\pi f_p(t) \quad \text{i.e.} \quad \phi_p(t) = \phi_p(0) + 2\pi \int_0^t f_p(u)\ du \tag{3.3}$$

The functions f_p, a_p, and ϕ_p are the frequency, amplitude, and phase of the p-th partial, respectively. The initial phase $\phi_p(0)$ will be ignored during analysis and set to an arbitrary value for resynthesis. This is a consequence of psychoacoustic experiments (see Chapter 1).

The choice of the $\pi/2$ value has been done in our *ReSpect* synthesis software package (see Chapter 4).

Although it is often useful to distinguish two types of partials – harmonic and non-harmonic partials – depending on whether or not they have equally spaced frequencies [MP80a, MP80b], this distinction will not be done here. This distinction is of great interest for pitch detection, and pitch-synchronous analysis methods can take advantage of it. However, this is not the case of the techniques presented in this chapter.

Another restriction is that the partials have to be spaced enough in frequency. Given any sound a, there must exist a minimal distance $d > 0$ so that:

$$\min_{i \neq j, t}\{|f_j(t) - f_i(t)|\} > d$$

This condition, which also prevents two partials frequencies from "crossing", is a reasonable hypothesis verified for many monophonic natural sounds. The reasons why it is needed will be discussed in Sections 3.2 and 3.3.

Section 3.1 presents the classic Fourier analysis and points out its main limitations and imprecisions, while Section 3.2 explains some interesting improvements to this classic Fourier analysis. In Section 3.3 we present our high precision Fourier analysis method using signal derivatives. After this short-time analysis, a partial-tracking stage is necessary. Section 3.4 describes the most famous partial-tracking strategies. Section 3.5 shows how we structure the partials in order to get the four parameters of the Structured Additive Synthesis (SAS) model (see Chapter 2). Short-time analysis, partial tracking, as well as SAS structuring are implemented in our *InSpect* analysis program, presented in Section 3.6. We have also implemented in *InSpect* an interesting technique for lossless compression of the sinusoidal modeling parameters. This technique is presented in Section 3.7. Finally, Section 3.8 shows how the reanalysis of the parameters coming from initial analysis could turn out to be extremely useful not only to enhance the compression ratio, but also to perform very interesting musical processing on the tremolo or vibrato of the sounds for example. This reanalysis turns out to be of great interest for pitch tracking and source separation too.

3.1 Fourier Analysis

Many sinusoidal models have been proposed in recent years, and have demonstrated their practical interest in software implementations like Lemur [FH96], SMS [Ser97b], and *ProSpect* (see Chapter 2), thanks to the always increasing power of modern computers. However, sinusoidal models are derived from the work of Joseph Fourier during the nineteenth century.

The Fourier transform (FT) and its discrete version – the discrete Fourier transform (DFT) – are mathematical operations defined by Equations 3.4 and 3.5, respectively:

$$\mathrm{FT}(f) = \int_{-\infty}^{+\infty} a(t)\ e^{-j2\pi f t}\ dt \tag{3.4}$$

$$\mathrm{DFT}[m] = \frac{2}{N}\sum_{n=0}^{N-1} a[n]\ e^{-j\frac{2\pi}{N}nm} \tag{3.5}$$

The Fast Fourier Transform (FFT) algorithm computes the discrete Fourier transform with a complexity of only $O(N\log(N))$. Note that in Equation 3.5 a $\frac{2}{N}$ factor has been added for the purposes of normalization (see Chapter 1).

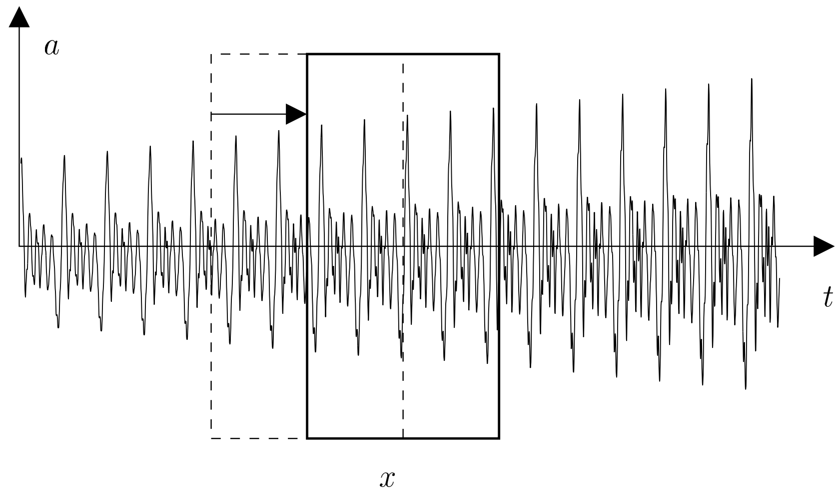


Figure 3.1: A sliding analysis window hops through the entire signal. At each step a frame window x of temporal signal is produced.

The Fourier transform converts the temporal signal (amplitude versus time) into a spectral representation (amplitude versus frequency). It gives the frequency image of the whole sound, and the entire signal is averaged into a single spectrum. This spectrum coincides with our perception only for stationary sounds. Because most sounds evolve in time, sound analysis algorithms must produce time-varying results. Those algorithms generally repeat the same procedures on successive short sections of sound. This kind of processing is called short-time analysis.

3.1.1 Short-Time Fourier Analysis

Among the most famous short-time analyses is the short-time Fourier transform [All77], which gives a time-dependent version of the Fourier transform. It produces a series of short-term spectra taken on successive – often overlapping – temporal frames, which are small pieces of temporal signal. In practice this is a discrete signal resulting from uniform sampling with sampling rate F_s.

A sliding analysis window hops through the entire signal, as shown in Figure 3.1. Usually the width N of the analysis window as well as the hop size are constants. Then, for each frame window x of N consecutive samples, its (discrete) Fourier transform X is computed:

$$X[m] = \frac{2}{N} \sum_{n=0}^{N-1} x[n]\ e^{-j\frac{2\pi}{N}nm} \tag{3.6}$$

The classic approach to estimate the frequency and amplitude of the partials is to scan the spectrum for local maxima – peaks – in the magnitude spectrum, and to determine the frequency, amplitude, and possibly phase at these maxima.

In fact a (discrete) windowing function w is always applied (multiplied) to x prior to the Fourier transform. Figure 3.2 illustrates this. Since $e^{j\phi} = \cos(\phi) + j\sin(\phi)$, the real and imaginary parts of the complex exponential function in Equation 3.6 are cosine and sine functions, respectively. Another way to understand Equation 3.6 is to look at Figure 3.2 from

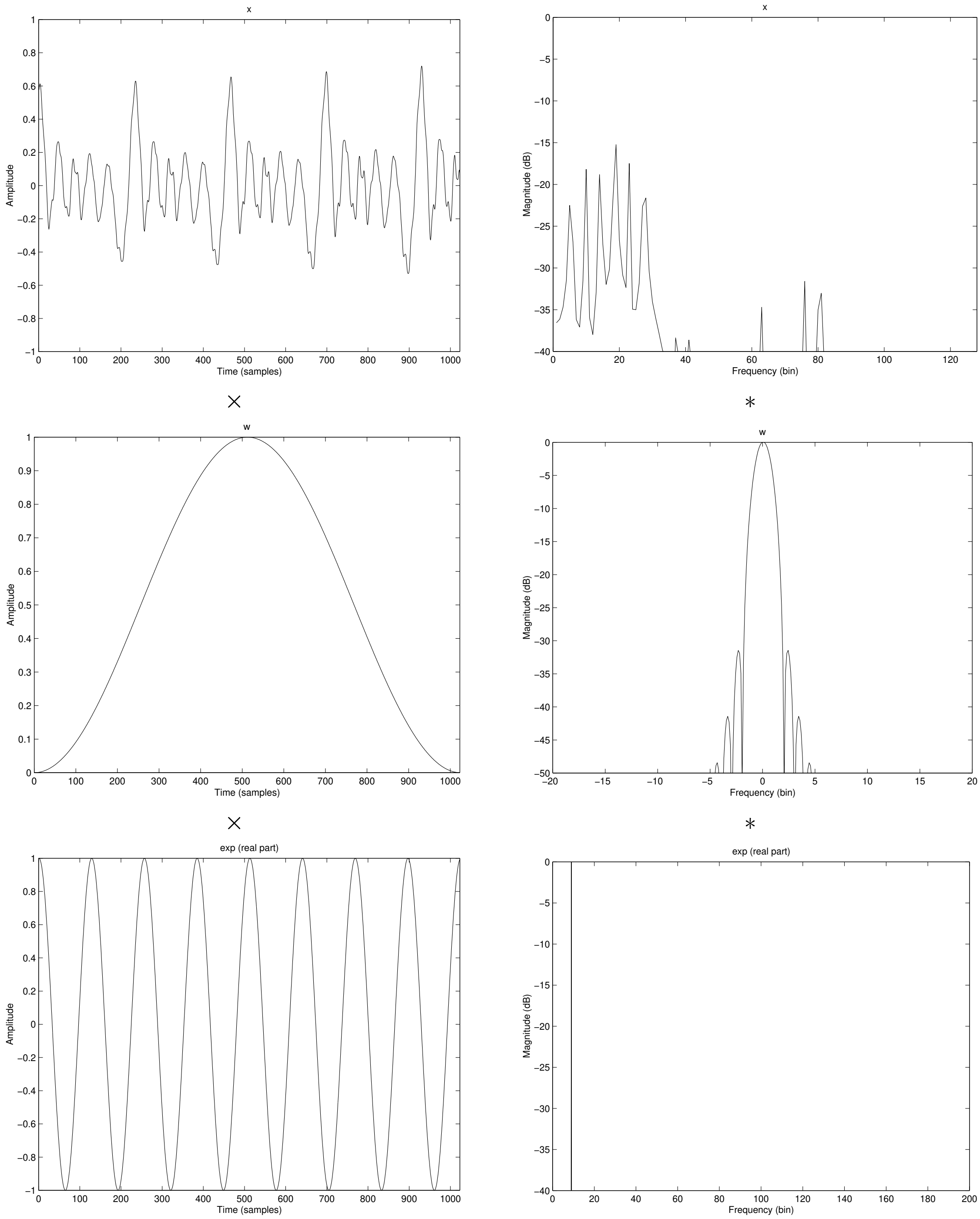


Figure 3.2: In Equation 3.6, an analysis window w is multiplied to the temporal frame x, then the result is multiplied by complex exponential functions. From top to bottom, the window frame x, the analysis window w, and the real part of one of the exponentials are displayed in both the time (left) and frequency (right) domains.

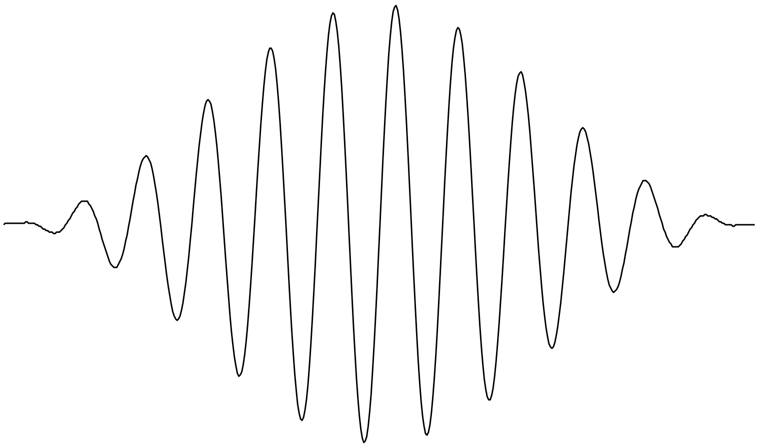


Figure 3.3: Example of a Gabor grain.

bottom to top and to consider that these cosines and sines are multiplied by the analysis window w, and that these products are then multiplied to the signal x. The products of the analysis window with oscillations look like wavelets, but are in fact Gabor grains (provided that w is a Gaussian though), as the one shown in Figure 3.3. That is why this transform is also called the Gabor transform when w is a Gaussian [Arf91].

If $w[k]$ is 1 for every $0 \leq k < N$ (and 0 outside) – as in Equation 3.6 – w is called the rectangular window (because the shape of its graph looks like a rectangle), and this is probably the worst analysis window.

The remainder of this section will focus on the (short-term) power spectra obtained from the short-time Fourier transform.

3.1.2 Choosing the Analysis Window

The choice of the analysis window is very important. Although an exhaustive discussion about analysis windows is beyond this chapter, Figure 3.1 provides a short summary of the characteristics of the most common analysis windows. Basic knowledge of analysis windows is required to fully understand the discussion below. Information about analysis windows can be found for example in [Har78] and [OS89].

Equations 3.7, 3.8, 3.9, 3.10, and 3.11 are the equations of the most commonly used analysis windows, that is the rectangular, Bartlett, Hamming, Hann, and Blackman windows, respectively. All these equations are valid only for $0 \leq n < N$. Outside this interval their value is simply 0.

$$
\begin{aligned}
w_{\text{rectangular},N}(n) &= 1 && (3.7)\\
w_{\text{Bartlett},N}(n) &= \begin{cases} \frac{2n}{N-1} & \text{if} \quad 0 \leq n \leq \frac{N-1}{2} \\ 2 - \frac{2n}{N-1} & \text{if} \quad \frac{N-1}{2} < n < N \end{cases} && (3.8)\\
w_{\text{Hamming},N}(n) &= 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right) && (3.9)\\
w_{\text{Hann},N}(n) &= \frac{1}{2}\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right) && (3.10)\\
w_{\text{Blackman},N}(n) &= 0.42 - 0.5\cos\left(\frac{2\pi n}{N-1}\right) + 0.08\cos\left(\frac{4\pi n}{N-1}\right) && (3.11)
\end{aligned}
$$

The first one is called the rectangular window because the shape of its graph in the temporal domain looks like a rectangle. Figure 3.4 shows the temporal representations of the other windows. The Bartlett window is also called the triangular window, again because of the shape of its graph. The Hann window is named after Julius von Hann, an Austrian meteorologist, and is often referred to as the Hanning window. The term "hanning" was used in [BT58] by Blackman and Tukey to describe the operation of applying this window to a signal and has since become the most widely used name for the window. But the "hanning" term was used mainly in an ironical way. The Hann window is close to the Hamming window in the frequency domain and close to the Blackman window in the time domain. However the Hann window has indeed very interesting properties, and turns out to be very useful for sound analysis and synthesis. That is why in my opinion the ironical term "hanning" should not be used in the remainder of this chapter. Figure 3.5 shows the power spectra of these analysis windows. Note that although the Hann and Blackman windows may look similar in the temporal domain, they have in fact very different spectra, and thus different properties.

Of course there are many other windows, such as the Kaiser windows [OS89] which allow to quantify the trade-off between the main-lobe width and side-lobe area. The Kaiser windows are defined by the following equation:

$$
w_{\text{Kaiser},\beta,N}(n) = \frac{I_0\left(\beta\sqrt{1 - \left(\frac{n-\alpha}{\alpha}\right)^2}\right)}{I_0(\beta)} \qquad (3.12)
$$

where I_0 is the modified Bessel function of the first kind and order 0. By tuning the β parameter, it is possible to design a Kaiser window with the same properties than those of the usual windows described above. Let A_{sl} be the ratio in dB of the amplitude of the main lobe (ml) to the amplitude of the largest side lobe (sl). The values for the classic analysis windows are given in Table 3.1. Let Δ_{ml} be the main-lobe width, defined as the symmetric distance between the central zero crossings in the frequency domain. To obtain this parameter for the classic analysis windows, just use the right column of Table 3.1 and multiply it by $\frac{2\pi}{F_s}$. The Kaiser and Schafer formulas can be used to determine the two β and N parameters from A_{sl}

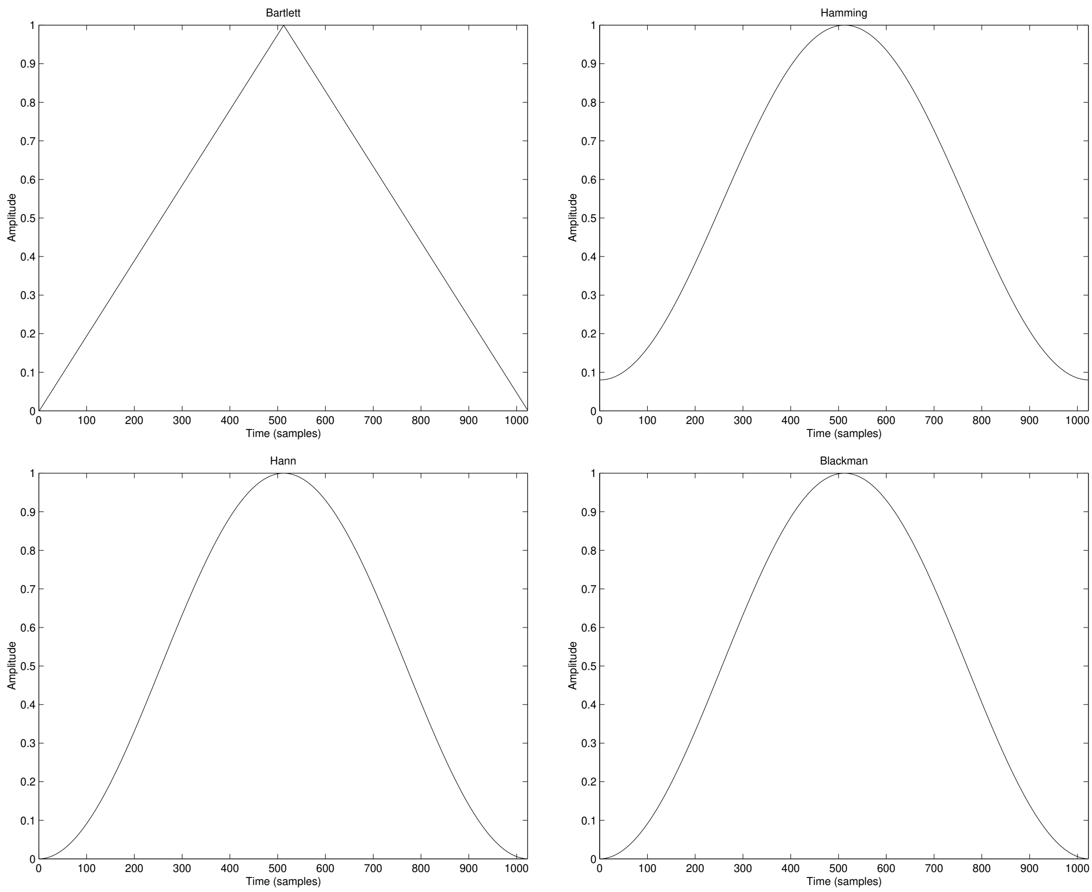


Figure 3.4: Temporal representations of some classic analysis windows.

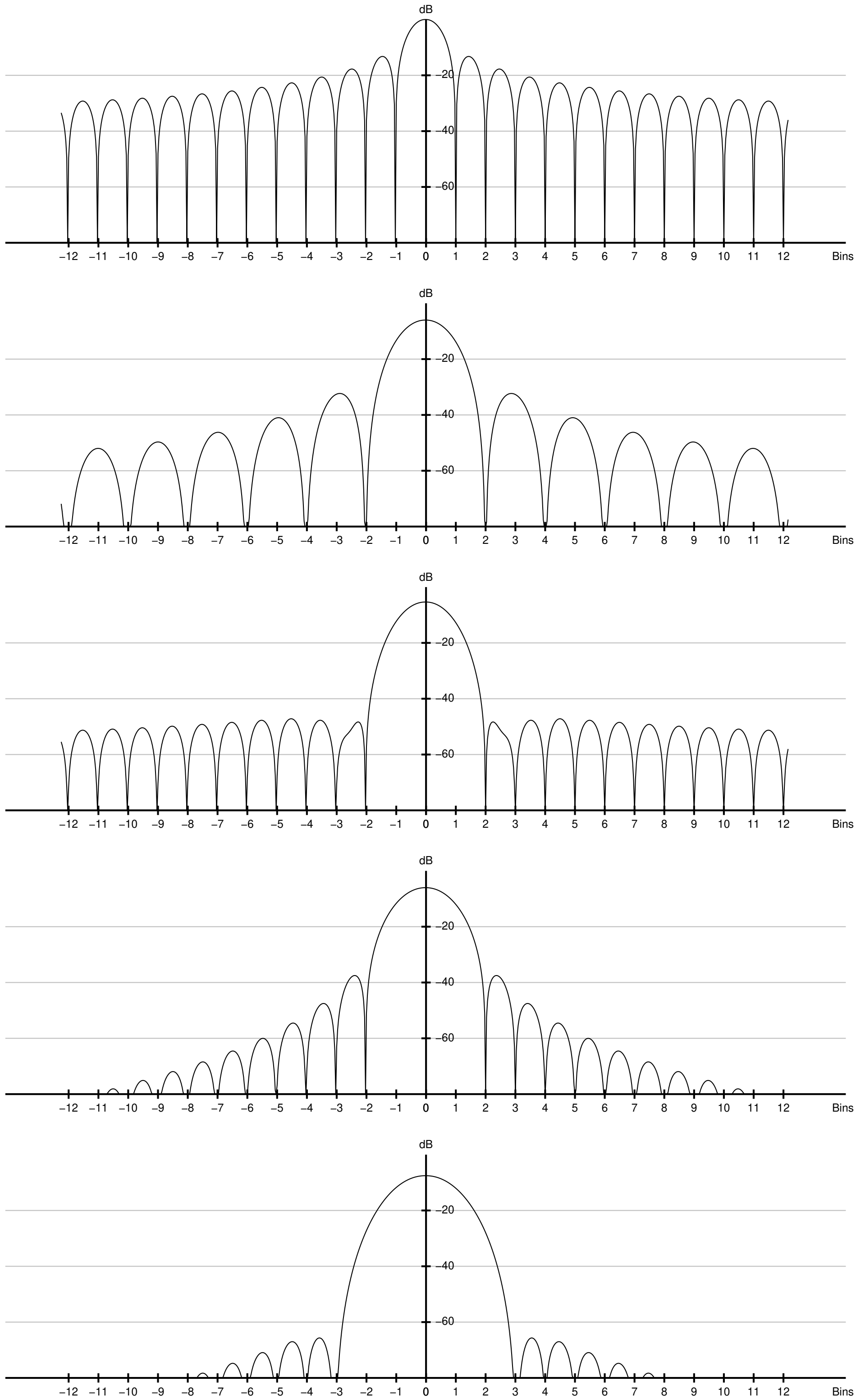


Figure 3.5: Power spectra of some analysis windows, from top to bottom: $W_{\mathrm{rectangular}}$, W_{Bartlett}, W_{Hamming}, W_{Hann}, and W_{Blackman}.

type	one-peak	side/main-lobe ratio ($-A_{\text{sl}}$)	main-lobe width
rectangular	yes	-13 dB	$2\mathcal{B}$
Bartlett (triangular)	no	-26 dB	$4\mathcal{B}$
Hann	yes	-31 dB	$4\mathcal{B}$
Hamming	no	-42 dB	$4\mathcal{B}$
Blackman	no	-58 dB	$6\mathcal{B}$

Table 3.1: Some analysis windows characteristics. The side/main-lobe ratio is the relative amplitude between the main lobe and the strongest side lobe. The main-lobe width is defined as the symmetric distance between the central zero crossings in the frequency domain.

and Δ_{ml}:

$$\beta(A_{\text{sl}}) = \begin{cases} 0 & \text{if} \quad A_{\text{sl}} \leq 13.26 \\ 13.26 + 0.7660(A_{\text{sl}} - 13.26)^{0.4} + \cdots & \\ \cdots + 0.09834\,(A_{\text{sl}} - 13.26) & \text{if} \quad 13.26 < A_{\text{sl}} \leq 60 \\ 0.12438\,(A_{\text{sl}} + 6.3) & \text{if} \quad 60 < A_{\text{sl}} < 120 \end{cases} \tag{3.13}$$

$$N(A_{\text{sl}}, \Delta_{\text{ml}}) = \left\lceil \frac{24\pi\,(A_{\text{sl}} + 12)}{155\Delta_{\text{ml}}} \right\rceil + 1 \tag{3.14}$$

Window Type

First, the type of the analysis window must be fixed, according to three principal requirements. First of all, a pure sinusoidal signal should always lead to a single local maximum in the magnitude spectrum, in order to have only one peak during the partial-tracking stage (see Section 3.4). More formally, with $\mathcal{B} = \frac{F_s}{N}$, for all $0 \leq f < \mathcal{B}$, the $(W(f + k\mathcal{B}))_{k \geq 0}$ sequence should be decreasing. Among all the windows that are considered in Figure 3.1, only the rectangular and Hann windows have this "one-peak" property. In fact this requirement is important only for the partial-tracking stage following the short-term analysis, and not for the analysis itself. This requirement can be weakened by considering only the maxima above a certain magnitude threshold. The next requirements are more important since they prevent neighboring frequency components from contaminating the measurement of any particular frequency component. The second requirement is that the side lobes can be neglected in comparison to the main lobe. This is the case for the Hann window, but not for the rectangular one. Then, the third requirement is that the main lobe must be narrow, in order to have good frequency resolution. But the price to pay for this is to increase the magnitudes of the side lobes. A compromise must be reached. Again the Hann window turns out to be a good candidate. The *InSpect* (see Section 3.6) analysis software implements many windows, and the Hann window has proven to perform very well in practice (see below).

Window Size

The size of the analysis window must then be precised. On the one hand N has to be large enough so that $\mathcal{B} < \min_{i \neq j}(|f_j - f_i|)$, which is always possible because of the model restrictions assumed at the beginning of this chapter. The reason behind this condition is that two frequencies must lie in different Fourier transform bins, because $\frac{f_i a_i + f_j a_j}{a_i + a_j} \neq f_i + f_j$ and this would cause problems with Equation 3.37 (bin contamination). On the other hand, the analysis window should remain small enough, so that the frequencies and amplitudes may not vary too much throughout the window.

3.1.3 Limitations of the Classic Fourier Analysis

The analysis window has a great impact on the analysis precision in both frequency and amplitude. Its effects on the spectrum must be reduced as much as possible. Figure 3.6 provides a clear synthetic view of two phenomena. It shows the result of the short-time Fourier transform on a single sinusoidal oscillator whose frequency is linearly increasing while its amplitude remains constant. The analyzed frequency curve is not a line as it should be, but a sort of stairs, due to spectrum sampling (frequency imprecision). And the analyzed amplitude curve is not flat, i.e. not a constant, but a succession of bumps, due to the shape of the analysis window main lobe (amplitude imprecision). These are the reasons why the classic phase vocoder can perform poorly when analyzing sounds with vibrato or tremolo, respectively.

Frequency Imprecision

Let us denote by S the spectrum of a signal s, and by $S(f)$ its value at frequency f. Since in practice the discrete Fourier transform is used, this spectrum is sampled from 0 Hz (DC component) to $\frac{F_s}{2}$ Hz (Nyquist frequency) by steps of $\frac{F_s}{N}$ Hz. Let us denote by $S[k]$ its value at frequency $k\mathcal{B}$ where $\mathcal{B} = \frac{F_s}{N}$. The Fourier transform can be regarded as a fixed filter bank, and the instantaneous amplitude and frequency are computed for each of the N channel filters or bins.

The first limitation comes from the fact that the frequency precision is inversely proportional to N. A good frequency precision requires a small $\mathcal{B}$, i.e. a large N, which leads to a poor precision in time (since N is the analysis window width in samples). On the contrary, a good time resolution leads to a poor frequency precision. This is the well-known trade-off of time versus frequency in the classic Fourier analysis.

Amplitude Imprecision

The spectrum of the wx product is the convolution $W * X$. If x is a pure sinusoidal oscillation with amplitude 1 and frequency f, X is a spectral ray: $|X(f)| = 1$ and $|X(f')| = 0$ for $f' \neq f$. But since W is similar to the window spectra shown in Figure 3.5, $W * X$ is far from being a spectral ray as it should be. More precisely, to obtain a spectral ray W should be as close to an impulse ($|W(0)| = 1$ and $|W(f)| = 0$ for $f \neq 0$) as possible, which is impossible with a finite-duration analysis window. So in practice W often consists of "lobes", and it is only required that the side-lobe amplitudes are negligible compared to the main-lobe amplitude. Unfortunately, this side/main-lobe ratio is inversely proportional to the main-lobe width, which determines the frequency resolution of the window. A good frequency resolution requires a small main-lobe width, but because of the shape of the lobe, the magnitude of the spectrum is distorted as shown in Figure 3.6. This is the reason why the amplitude of a sinusoidal oscillator is so distorted as its frequency goes from bin to bin (see Figure 3.7). Any good analysis method should take care of this phenomenon. Of course such little deformations cannot generally be heard, but they may become dramatically audible as soon as some transformations are performed on the sound. Figure 3.2 shows that the amplitude imprecisions for the most common analysis windows are quite important. In order to minimize

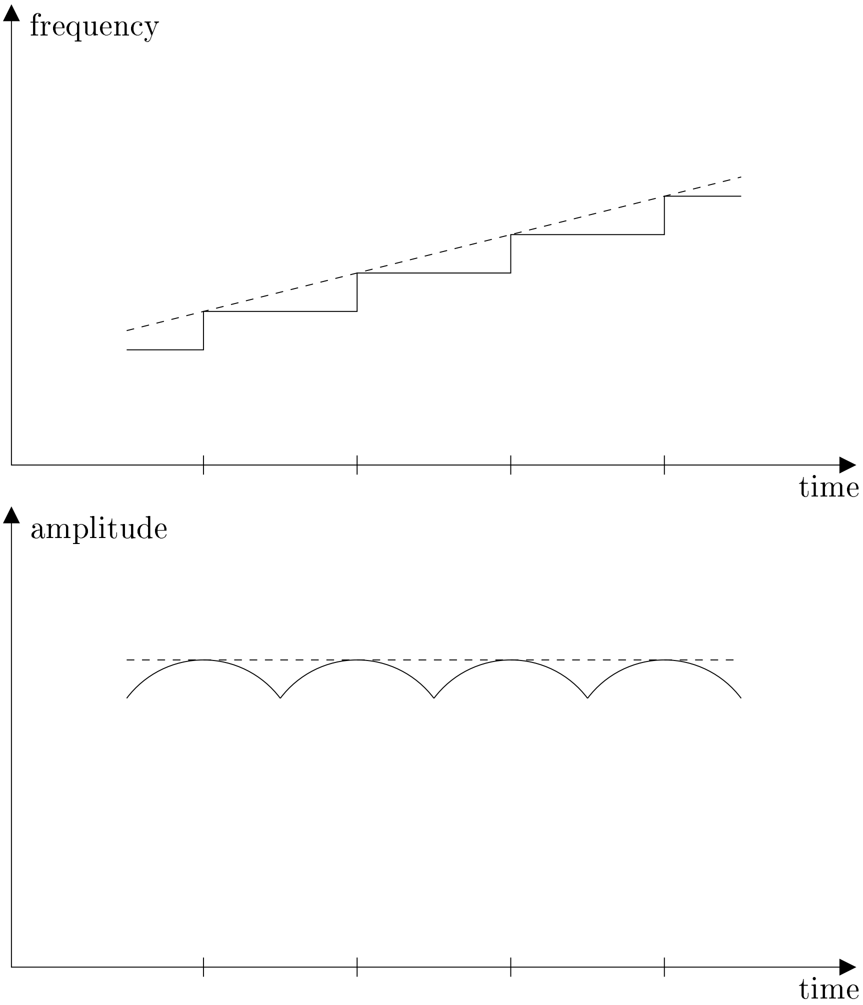


Figure 3.6: Original (dashed) versus Fourier analyzed (solid) frequency and amplitude evolutions for a single sinusoidal oscillator whose frequency is linearly increasing while its amplitude remains constant. (The marks on the time axis indicate when the oscillator frequency goes from one bin to the other.)

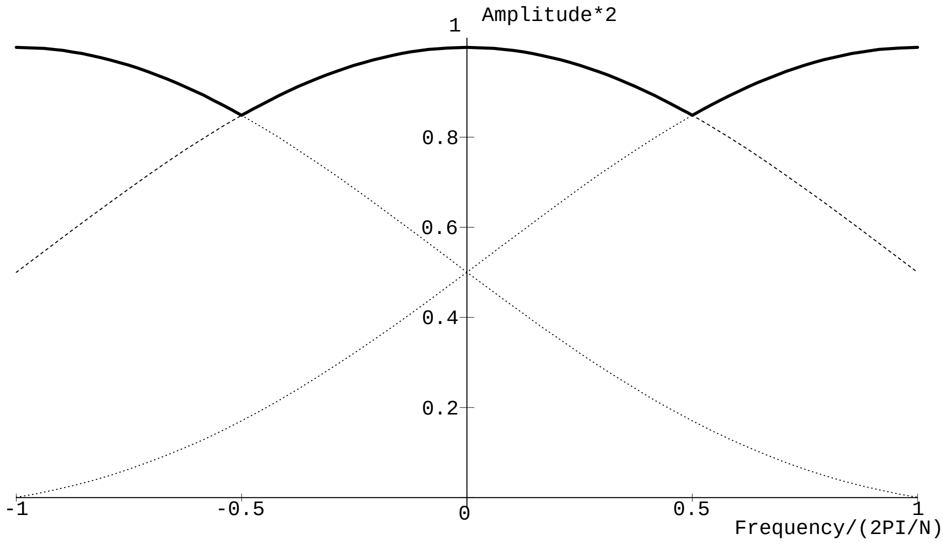


Figure 3.7: Hann window lobes.

rectangular	Bartlett (triangular)	Hann	Hamming	Blackman
36%	19%	15%	18%	12%

Table 3.2: Amplitude imprecision for some windows.

this imprecision, a "flat" main lobe is necessary. But this is possible only with a very large main lobe. So there is a sort of trade-off of amplitude precision versus frequency precision.

3.2 Improvements to Fourier Analysis

Accurate short-time spectral analysis is extremely important for many computer music analysis / synthesis methods, such as the George-Smith [GS90] analysis, as well as the McAulay-Quatieri analysis [MQ86] where accurately identifying frequency components is the first step for a key technique called partial tracking (see Section 3.4). When no precautions are taken, the imprecisions pointed out in the previous section appear. As a consequence, many analysis / synthesis software tools suffer from the short-time Fourier analysis limitations shown above. Many techniques have been proposed in order to reduce these limitations.

In spite of its many drawbacks, the short-time Fourier transform is often used in the very first step of the analysis process. Regarding frequency, the imprecision in frequency Δ_f of the classic short-time Fourier analysis or the Gabor analysis is a constant, inversely proportional to the analysis window width N.

We need to extract the partials of the sound, that is to measure for each partial its frequency, amplitude, and possibly phase. This section focuses on methods that reduce the imprecision in frequency, as well as the imprecision in amplitude and possibly phase.

Wavelets perform a frequency decomposition with a constant Δ_f/f factor, that is a frequency imprecision Δ_f proportional to the frequency f. This resolution is closer to the behavior of the auditory system. Moreover the wavelet transforms have associated inverse

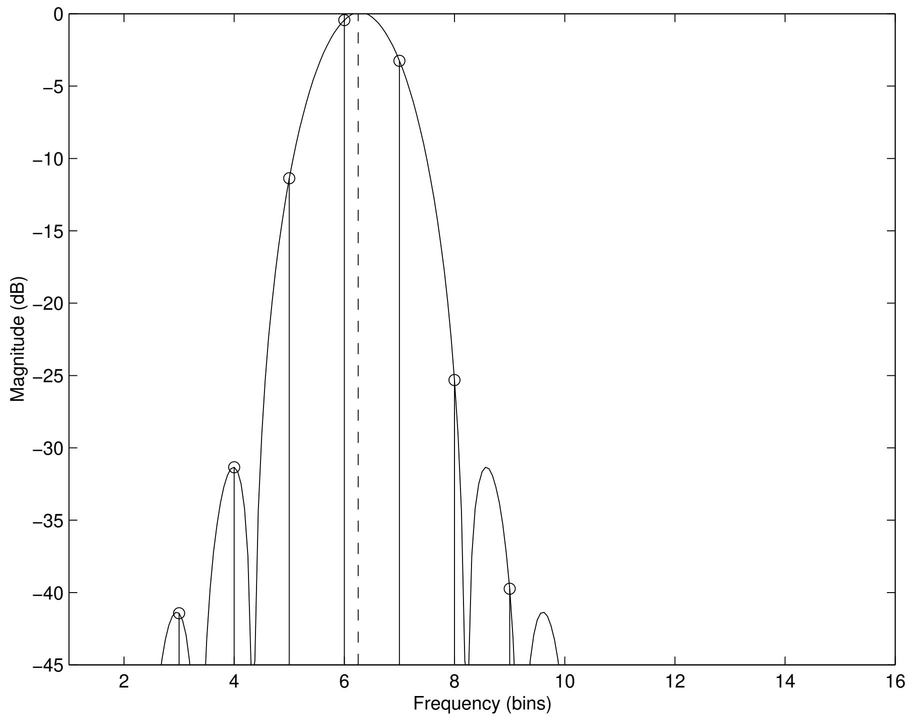


Figure 3.8: When the analyzed frequency is not a multiple of the lowest Fourier transform frequency, the neighbors of the main bin have significant magnitudes too.

transforms, thus facilitating the synthesis stage [Eva91].

The problem is that the parameters obtained from wavelet analysis are not well-suited for musical sound transformations at all. The harmonics of a complex sound must be identified in order to allow musical transformations without dramatically audible artifacts. Thus an important problem with the constant Δ_f/f factor in the wavelet transforms occurs for high-frequency harmonics, for which the frequency resolution is so bad that several harmonics may be averaged in one coefficient of the wavelet transform.

3.2.1 About Estimators

Partials are spectral peaks, and should correspond to local maxima in the power spectrum. To determine the parameters – frequency, amplitude, and phase – of the partials, the classic Fourier analysis considers only the Fourier transform bin with the greatest magnitude. This gives the exact parameters only if the frequency of the partial is exactly one of those used by the Fourier transform, which is very unlikely in practice. When the frequency to analyze lies between two Fourier transform frequencies, the convolution of this frequency with the analysis window is sampled by the discrete Fourier transform. The effect on the spectrum is that there is a local maximum for a certain Fourier transform bin, but this time the neighboring bins have significant magnitudes too, as shown in Figure 3.8. In order to increase the precision of the classic Fourier analysis for the measurement of a certain spectral peak, a good idea is to consider not only this bin, but also some of its neighbors.

Estimators are algorithms using the main bin and some of its left and right neighbors in order to refine the measurement of the parameters of the partial. Spectral components

Barycentric Peak Interpolation
$a_l \leftarrow \lvert X[m_p - 1]\rvert$
$a_c \leftarrow \lvert X[m_p]\rvert$
$a_r \leftarrow \lvert X[m_p + 1]\rvert$
$d \leftarrow (a_l - a_r)/(a_l + a_c + a_r)$
$m_p \leftarrow m_p + d$

Table 3.3: Algorithm for barycentric peak interpolation.

Quinn's First Estimator
$a_+ \leftarrow (X[m_p+1].re \cdot X[m_p].re + X[m_p+1].im \cdot X[m_p].im)/\lvert X[m_p]\rvert^2$
$d_+ \leftarrow a_+/(a_+ - 1)$
$a_- \leftarrow (X[m_p-1].re \cdot X[m_p].re + X[m_p-1].im \cdot X[m_p].im)/\lvert X[m_p]\rvert^2$
$d_- \leftarrow a_-/(1 - a_-)$
$d \leftarrow ((d_+ > 0)\ \&\&\ (d_- > 0))\ ?\ d_+\ :\ d_-$
$m_p \leftarrow m_p + d$
Quinn's Second Estimator
$\tau(x) = \frac{1}{4}\log(3x^2 + 6x + 1) - \frac{\sqrt{6}}{24}\log\left((x + 1 - \sqrt{\frac{2}{3}})/(x + 1 + \sqrt{\frac{2}{3}})\right)$
$a_+ \leftarrow (X[m_p+1].re \cdot X[m_p].re + X[m_p+1].im \cdot X[m_p].im)/\lvert X[m_p]\rvert^2$
$d_+ \leftarrow a_+/(a_+ - 1)$
$a_- \leftarrow (X[m_p-1].re \cdot X[m_p].re + X[m_p-1].im \cdot X[m_p].im)/\lvert X[m_p]\rvert^2$
$d_- \leftarrow a_-/(1 - a_-)$
$d \leftarrow (d_+ + d_-)/2 + \tau(d_+{}^2) - \tau(d_-{}^2)$
$m_p \leftarrow m_p + d$

Table 3.4: Algorithms for Quinn's first (top) and second (bottom) estimators.

which are close in frequency additively interfere, affecting each-other's spectra. It is therefore desirable to make all spectral measurements close to the maximum of a peak, so as to maximize the influence from that peak and minimize the influence from adjacent peaks. That is the reason why in practice only the left and right neighbors are used, to prevent other partials – close in frequency – from interfering in the measurement.

There are many estimators. Jacobsen [Jac00] and Kootsookos [Koo99] have taken an inventory of them. Among these estimators are the barycentric peak interpolation, the two Quinn's estimators, as well as the Jain and Grandke methods. Their algorithms can be found for example in [Don99] and are given in Tables 3.3, 3.4, and 3.5, respectively.

Unfortunately it appears that most of these estimators are mostly empirical, that is without any clear mathematical justification. That is the case for the barycentric interpolation, as well as the Jain and Grandke methods. However Quinn published the justifications for his estimators in [Qui94].

All these estimators have been implemented in the new version of our analysis tool *InSpect* (see Section 3.6). It appears that the accuracies of these estimators are not satisfactory, even if Quinn's second estimator seems much better than the others.

Other estimators exist, such as the parabolic peak interpolation and the triangular algo-

Jain Method	**Grandke Method**
$a_l \leftarrow \|X[m_p-1]\|$	$a_l \leftarrow \|X[m_p-1]\|$
$a_c \leftarrow \|X[m_p]\|$	$a_c \leftarrow \|X[m_p]\|$
$a_r \leftarrow \|X[m_p+1]\|$	$a_r \leftarrow \|X[m_p+1]\|$
if $(a_l > a_r)$	if $(a_l > a_r)$
$\quad a \leftarrow \frac{a_c}{a_l}$	$\quad a \leftarrow \frac{a_c}{a_l}$
$\quad d \leftarrow \frac{a}{1+a}$	$\quad d \leftarrow \frac{2a-1}{a+1}$
$\quad m_p \leftarrow m_p - 1 + d$	$\quad m_p \leftarrow m_p - 2 + d$
else	else
$\quad a \leftarrow \frac{a_r}{a_c}$	$\quad a \leftarrow \frac{a_r}{a_c}$
$\quad d \leftarrow \frac{a}{1+a}$	$\quad d \leftarrow \frac{2a-1}{a+1}$
$\quad m_p \leftarrow m_p + d$	$\quad m_p \leftarrow m_p - 1 + d$

Table 3.5: Algorithms for Jain (left) and Grandke (right) methods.

Parabolic Peak Interpolation
`/* frequency correction */`
$a_l \leftarrow 20\ \log_{10}(\|X[m_p-1]\|)$
$a_c \leftarrow 20\ \log_{10}(\|X[m_p]\|)$
$a_r \leftarrow 20\ \log_{10}(\|X[m_p+1]\|)$
$d \leftarrow \frac{1}{2}\frac{a_l-a_r}{a_l-2a_c+a_r}$ `/* Brent */`
$m_p \leftarrow m_p + d$
`/* amplitude (in dB) correction */`
$a_c \leftarrow a_c - (a_l - a_r) \cdot d/4$

Table 3.6: Algorithm for parabolic peak interpolation, also known as quadratic interpolation.

rithm. These two estimators have solid mathematical foundations and perform extremely well in practice.

3.2.2 Parabolic Interpolation

The shape of the main lobe of most analysis windows looks like a parabola, as shown in Figure 3.8 and on the zoom on the main lobe in Figure 3.9. For each partial, the parabolic interpolation uses the main bin and its left and right neighbors in a curve-fitting process with a parabola, using the Brent method [PTVF92] to estimate the maximum of the parabola. Table 3.6 gives the resulting algorithm.

This method gives best results with a Gaussian analysis window, because the shape of its main – and unique – lobe is exactly a parabola in the power spectrum (see Figure 3.10). Gaussian-like windows can be calculated using the following equation:

$$w_{\mathrm{Gauss},\alpha,N}(n) = e^{-\alpha\left(\frac{2n}{N}-1\right)^2} \quad \text{for} \quad 0 \leq n < N \tag{3.15}$$

In practice, we use $\alpha = 3$.

In order to increase the precision of the parabolic interpolation, zero-padding can be used as in SMS [Ser89, Ser97b].

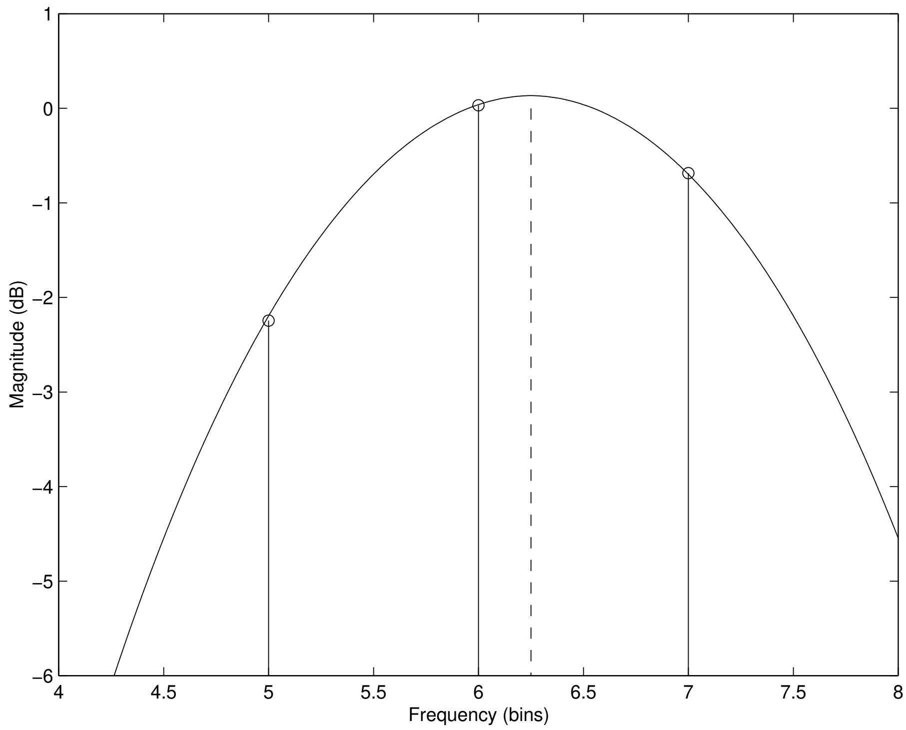


Figure 3.9: The main peak and its two neighbors are located on a parabola corresponding to the shape of the main lobe of the analysis window.

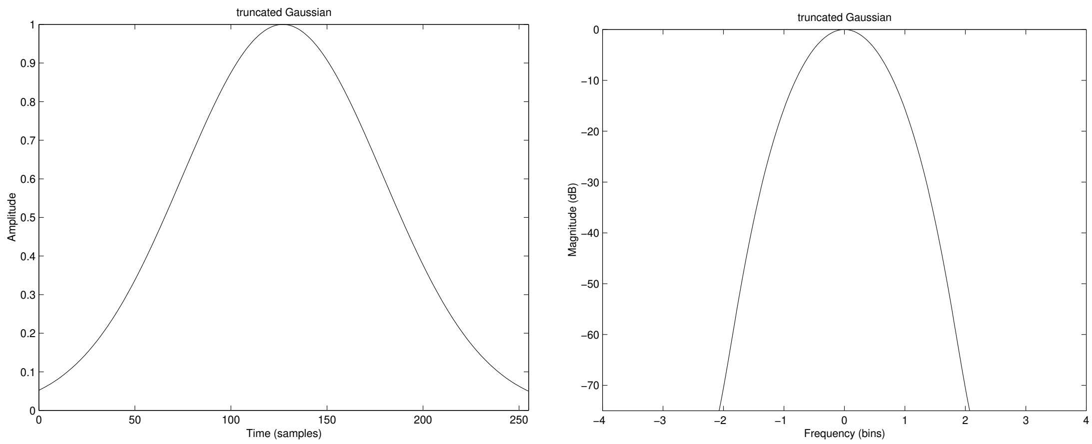


Figure 3.10: The truncated Gaussian window of Equation 3.15 with $N = 256$ (left) and its power spectrum (right), showing a parabolic main lobe.

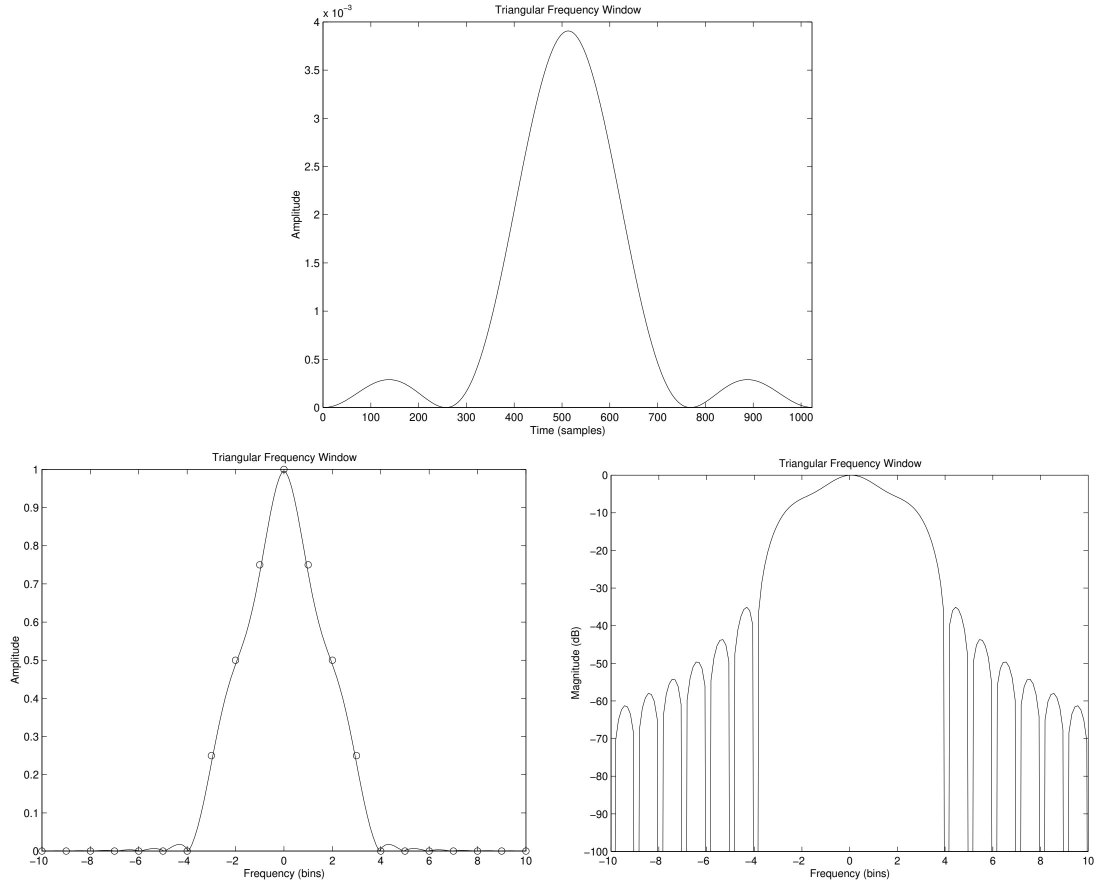


Figure 3.11: Triangular Frequency (TriFreq) window ($S = 4$, $N = 1024$) in the time (top) and frequency (bottom) domains. Left and right plots correspond, respectively, to the linear and logarithmic (dB) scales for the magnitude.

3.2.3 Triangular Algorithm

The triangle analysis algorithm [AKZ99, KZ00] – also known as the triangular algorithm – is named after the shape of the analysis window in the frequency domain. Keiler and Zölzer propose to use a window whose magnitude response can be determined by a simple function. A triangular frequency response can be described by only two lines. After applying a Fourier transform to the input data, for each detected local maximum of the magnitude spectrum a fitting of two lines in the least mean square error sense is performed through the spectrum data. This is the same idea as the parabolic interpolation, but this time the window is not a Gaussian, and the curve-fitting is performed using a "triangle" instead of a parabola. The triangular frequency (TriFreq) window is computed from its magnitude spectrum using the inverse Fourier transform. Figure 3.11 shows the TriFreq window for a triangle slope of $S = 4$.

3.2.4 Phase Distortion Analysis

A fundamental assumption when using the Fourier transform is that the partials are assumed to have constant frequency and amplitude throughout the analysis window. Since musical signal are in fact pseudo-periodic, this assumption is only a good approximation for short time windows. However the requirement for good frequency resolution often necessitates long time windows. Consequently the spectra produced by the Fourier transform contain artifacts which are due to the non-stationarities of the audio signals.

The Phase Distortion Analysis [MB95, MC98] is a technique introduced by Masri for extracting non-stationary elements from spectra produced by the Fourier transform, by making use of the artifacts they produce. In particular, linear frequency modulation and exponential amplitude modulation can be determined from the phase distortion that occurs around spectral peaks. In a practical situation the frequency and amplitude modulations will not follow such idealized trajectories as linear frequency modulation and exponential amplitude modulation. However the methodology can be used successfully and its estimations are largely accurate, when there is a presence of higher order modulation.

The Fourier transform of a windowed, stationary sinusoid is the Fourier transform of the window function, centered about the frequency of the sinusoid, and sampled at frequencies corresponding to the bins of the Fourier transform. It is also scaled according to the amplitude of the sinusoid, and rotated to the instantaneous phase of the sinusoid at the center of the time-window. Modulation of the frequency and/or amplitude of the sinusoid results in a widening of the spectral shape, distortion to its form (particularly around the main lobe), and phase distortion. However the frequency location, amplitude, and phase at the maximum of the main lobe are minimally affected, unless the distortion is severe.

The distortion is dependent on the window function but experiments on the classic windows – rectangular, Bartlett, Hamming, Hann, and Blackman – suggest that the form of the distortion is identical, and it is only the actual values that differ. In all cases, the measurements were found to be invariant of the frequency and amplitude of the modulated sinusoid. As a consequence, the distortion only depends on the modulation itself.

For an unmodulated sinusoid, the phase is constant across the main lobe as shown in Figure 3.13, provided that the zero-phase windowing technique is used. This technique – also used in SMS [Ser89, Ser97b] – consists in using an odd-length analysis window centered in a larger – zero-padded – Fourier transform buffer at the origin in order to obtain the phase spectrum free of the linear phase trend induced by the analysis window. Figure 3.12 illustrates this.

For sinusoids of linearly increasing or decreasing frequency, the phase either side of the maximum is, respectively, reduced or raised, as shown in Figure 3.14. The degree of phase distortion is dependent on the rate of change of frequency, and has to be calibrated for the method to take advantage of it. Whereas the phase distortion for linear frequency modulation is equal either side of the maximum, in the case of exponential amplitude modulation, the phase distortion is of equal magnitude but opposite sign, as shown in Figure 3.15. The phase distortions of linear amplitude and exponential frequency modulations are additive. At any offset from the maximum, in the range -1 to $+1$ bin, the total phase distortion is the sum of the distortions due to the linear amplitude and exponential frequency modulations. If two measurements are taken one either side of – and equidistant from – the maximum, then the amounts of distortion due to frequency and amplitude are, respectively, the sum and the difference both scaled by 0.5.

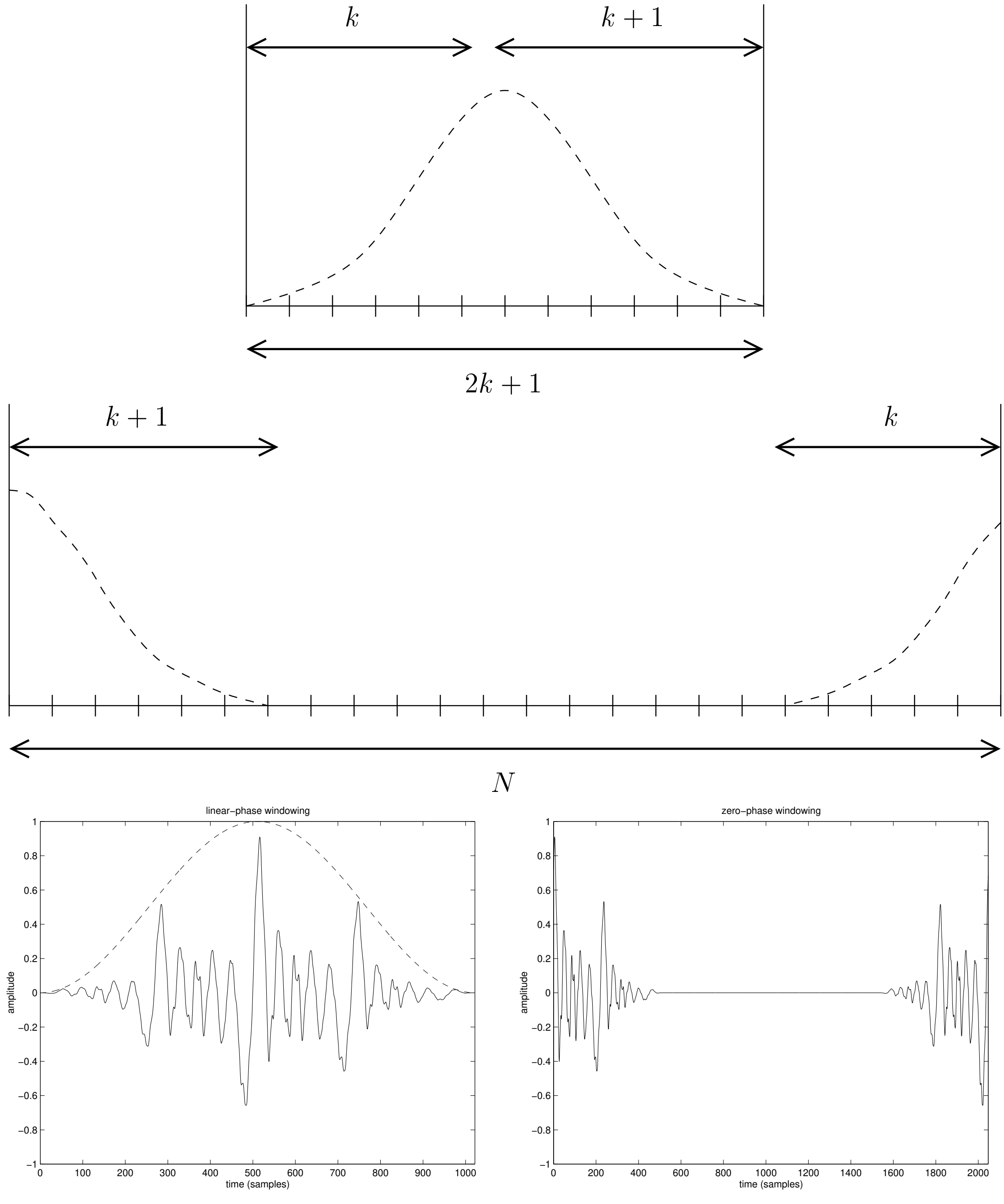


Figure 3.12: Linear-phase (top) versus zero-phase (middle) windowing, consisting in using an odd-length analysis window (width $2k+1$) centered in a larger – zero-padded – Fourier transform buffer (width N) at the origin in order to obtain the phase spectrum free of the linear phase trend induced by the analysis window. The same comparison is made during the analysis of a signal (bottom).

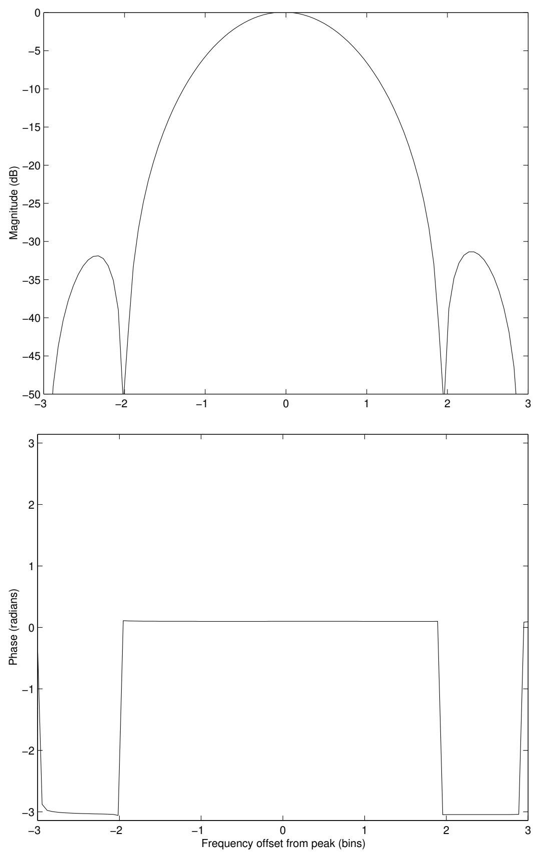


Figure 3.13: Magnitude and phase of an unmodulated sinusoid using zero-phase windowing (left) and the classic linear phase windowing (right).

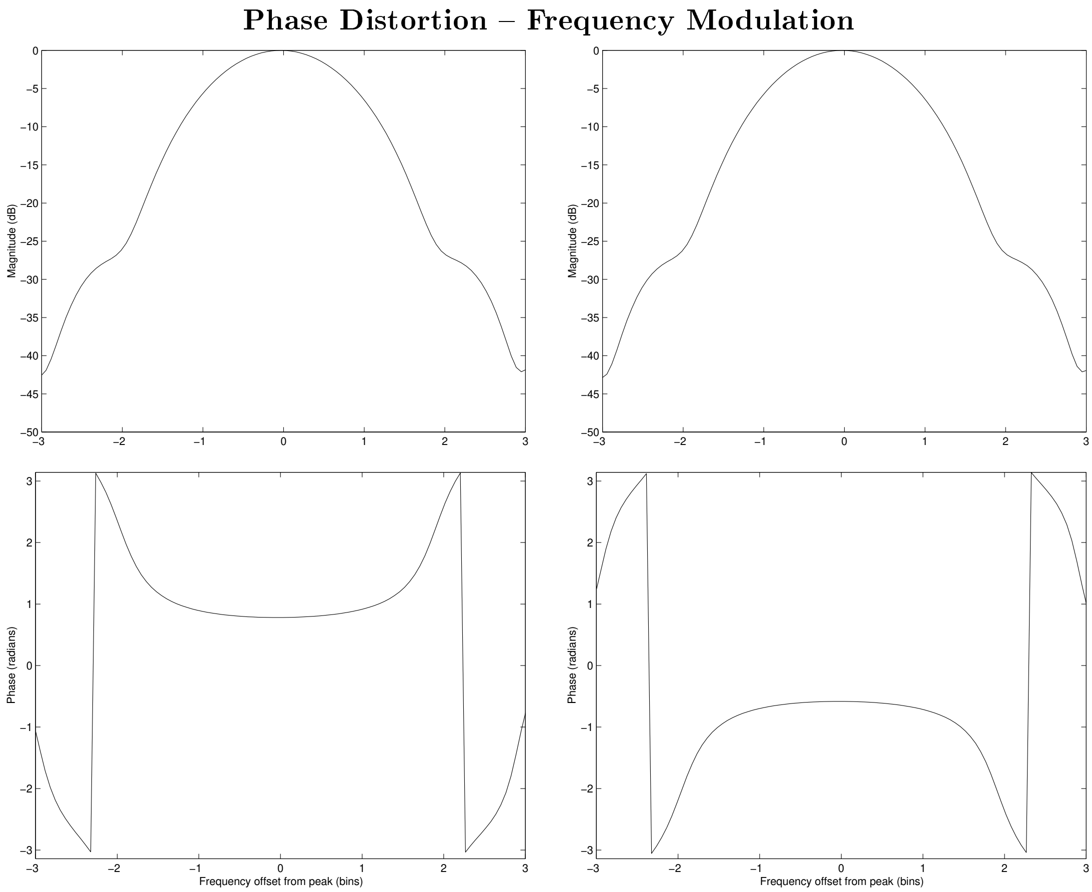


Figure 3.14: Magnitude and phase of a sinusoid modulated in frequency. The frequency of the sinusoid changes by -1 (left) and +1 (right) bin during the analyzed frame window.

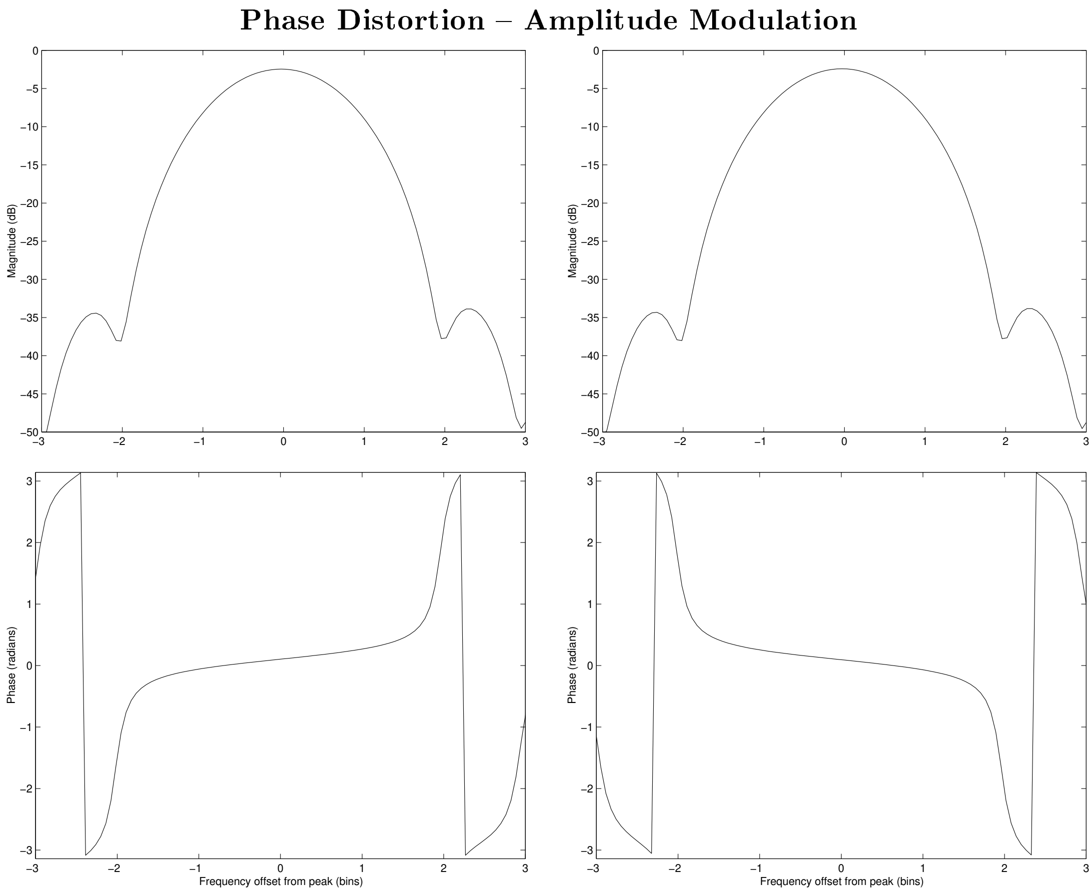


Figure 3.15: Magnitude and phase of a sinusoid modulated in amplitude. The frequency of the sinusoid changes by -6 dB (left) and +6 dB (right) during the analyzed frame window.

3.2.5 Spectrum Reassignment and Analytically Differentiated Windows

"Reassignment" has been proposed to improve time-frequency representations. In usual time-frequency representations, the values obtained when decomposing the signal on the time-frequency atoms are assigned to the geometrical center of the cells (center of the analysis window and bins of the Fourier transform). Auger and Flandrin propose in [AF95] to assign each value to the center of gravity of the cell's energy. The method uses the knowledge of the first derivative w' – obtained by analytical differentiation – of the analysis window w in order to adjust the frequency inside the Fourier transform bin. For example, if the analyzed frequency leads to a maximum of magnitude at the k-th bin of the spectrum of the Fourier transform, frequency reassignment is given by the following equation:

$$\omega_r(t,\omega_k) = \omega_k - \mathrm{Im}\left(\frac{\mathrm{STFT}_{w'}(\omega_k)}{\mathrm{STFT}_w(\omega_k)}\right) \tag{3.16}$$

where ω_k is the digital frequency (see Chapter 1) corresponding to bin number k – that is corresponding to frequency $f_k = kF_s/N$ – and ω_r is the reassigned digital frequency, corresponding to frequency $f_r = F_s\omega_r/2\pi$. STFT_w and $\mathrm{STFT}_{w'}$ are the short-time Fourier transforms of the signal using, respectively, w or its first derivative w' as the analysis window.

This method was recently implemented in SINOLA [PR99] by Peeters. Borum and Jensen also present in [BJ99] the use of a similar method for analysis / synthesis. We have implemented spectrum reassignment – among other analysis methods – in our *InSpect* analysis program (see Section 3.6).

In order to obtain Equation 3.16 (its continuous version, though), we can consider the continuous short-time Fourier transform of signal a at time t and frequency Ω and using the analysis window w. Its expression is given, for example, by the following equation:

$$\mathrm{STFT}_{a,w}(t,\Omega) = \int_{-\infty}^{+\infty} a(t+\theta)\ w(\theta)\ e^{-j\Omega\theta}\ d\theta \tag{3.17}$$

Introducing $\tau = t + \theta$ gives:

$$\mathrm{STFT}_{a,w}(t,\Omega) = \int_{-\infty}^{+\infty} a(\tau)\ w(\tau - t)\ e^{-j\Omega(\tau-t)}\ d\tau \tag{3.18}$$

The complex spectra resulting from this transform are:

$$\mathrm{STFT}_{a,w}(t,\Omega) = A_{a,w}(t,\Omega)\ e^{j\Phi_{a,w}(t,\Omega)} \tag{3.19}$$

Let us now focus on the "instantaneous frequency":

$$\Omega_{a,w}(t,\Omega) = \frac{d\Phi_{a,w}(t,\Omega)}{dt} \tag{3.20}$$

If we consider the imaginary part of the differentiation of the logarithm of the short-time Fourier transform, together with Equations 3.19 and 3.20, we have:

$$\begin{aligned}
\mathrm{Im}\left[\frac{d}{dt}\log(\mathrm{STFT}_{a,w}(t,\Omega))\right] &= \mathrm{Im}\left[\frac{d}{dt}\left(\log(A_{a,w}(t,\Omega) + j\Phi_{a,w}(t,\Omega)\right)\right] \\
\cdots &= \frac{d\Phi_{a,w}(t,\Omega)}{dt} \\
\cdots &= \Omega_{a,w}(t,\Omega)
\end{aligned}$$

If we now consider the same expression, but this time considering only Equation 3.18, we have:

$$\begin{aligned}
\mathrm{Im}\left[\frac{d}{dt}\log(\mathrm{STFT}_{a,w}(t,\Omega))\right] &= \mathrm{Im}\left[\frac{\frac{d}{dt}\left(\int_{-\infty}^{+\infty} a(\tau)\ w(\tau-t)\ e^{-j\Omega(\tau-t)}\ d\tau\right)}{\mathrm{STFT}_{a,w}(t,\Omega)}\right] \\
\cdots &= \mathrm{Im}\left[\frac{j\Omega\mathrm{STFT}_{a,w}(t,\Omega)-\mathrm{STFT}_{a,w'}(t,\Omega)}{\mathrm{STFT}_{a,w}(t,\Omega)}\right] \\
\cdots &= \Omega-\mathrm{Im}\left[\frac{\mathrm{STFT}_{a,w'}(t,\Omega)}{\mathrm{STFT}_{a,w}(t,\Omega)}\right]
\end{aligned}$$

Using the two previous results, we obtain the continuous version of Equation 3.16:

$$\Omega_{a,w}(t,\Omega)=\Omega-\mathrm{Im}\left[\frac{\mathrm{STFT}_{a,w'}(t,\Omega)}{\mathrm{STFT}_{a,w}(t,\Omega)}\right] \tag{3.21}$$

Considering the Signal Derivative instead of the Window Derivative
The previous equation involves the $a\ w$ and $a\ w'$ products, that is, respectively, the $A * W$ and $A * W'$ convolutions (using the uppercase notation for spectra introduced in Chapter 1). We could consider $a'\ w$ instead of $a\ w'$, that is the $A' * W$ instead of $A * W'$.

In practice the discrete versions of Equations 3.17 and 3.18 are not equivalent. Equation 3.18 is a phase-modulated version of Equation 3.17 [BJ99], and Equation 3.20 is not valid anymore if we consider Equation 3.17 instead of Equation 3.18. However, we have:

$$\begin{aligned}
\mathrm{Im}\left[\frac{d}{dt}\log(\mathrm{STFT}_{a,w}(t,\Omega))\right] &= \mathrm{Im}\left[\frac{\frac{d}{dt}\left(\int_{-\infty}^{+\infty} a(t+\theta)\ w(\theta)\ e^{-j\Omega\theta}\ d\theta\right)}{\mathrm{STFT}_{a,w}(t,\Omega)}\right] \\
\cdots &= \mathrm{Im}\left[\frac{\mathrm{STFT}_{a',w}(t,\Omega)}{\mathrm{STFT}_{a,w}(t,\Omega)}\right]
\end{aligned}$$

$$\text{but} \qquad \Omega_{a,w}(t,\Omega)\neq\mathrm{Im}\left[\frac{\mathrm{STFT}_{a',w}(t,\Omega)}{\mathrm{STFT}_{a,w}(t,\Omega)}\right] \tag{3.22}$$

But we will show in Section 3.3 that:

$$\Omega_{a,w}(t,\Omega)=\frac{|\mathrm{STFT}_{a',w}(t,\Omega)|}{|\mathrm{STFT}_{a,w}(t,\Omega)|} \tag{3.23}$$

3.2.6 Higher Order / High Resolution Spectral Analysis

The Fourier transform has been viewed traditionally as incapable of yielding more than a linear phase representation, even if the Phase Distortion Analysis [MB95, MC98] is capable of yielding second order phase information. As a result higher order phase representations, which can describe non-stationarities of frequency, have been – and continue to be – developed. These are largely based on the Wigner-Ville transform, which achieves quadratic phase – that is linear frequency modulation – representation.

For signals that are mono-component and non-stationary these higher order spectra have proven to be very useful. However for multi-component signals such as sounds – additively composed of sinusoids in the sound model considered in this chapter – the spectra display peaks not only for each component (the auto terms), but also for the correlation between components (the cross terms). The cross terms are often large enough to be indistinguishable from the auto terms, and can even mask them at times [BR92]. Current research is attempting to overcome this problem by developing techniques that suppress the cross terms [BZ92, OMQ92].

Anyway, the Wigner-Ville transform – as well as high-resolution spectral analysis in general – still requires a very important computational power, and its implementation is actually limited principally to parallel calculators...

3.3 High Precision Fourier Analysis using Signal Derivatives

We propose in [DCM98, Mar98, DCM00] the n-order (short-time) Fourier Transform (FT^n). This method is an enhancement of the classic short-time Fourier transform, providing greater accuracy for both frequency and amplitude with small analysis windows, thus permitting greater time resolution. The FT^n method is particularly well-suited for analyzing the deterministic component of sound.

Precisely, this method extends the classic short-time Fourier transform by also considering the signal derivatives, which effectively leads to efficient spectral parameter extraction, and thus allows precise modifications of the inner structures of the sounds. FT^n takes advantage of the first n signal derivatives in order to improve the precision of the Fourier analysis not only in frequency and amplitude but also in time, thus minimizing the problem of the trade-off of time versus frequency in the classic short-time Fourier transform. This method is of great interest for extracting spectral modeling parameters from existing sounds.

After a detailed theoretical presentation of the enhanced analysis method, the way to implement its discrete version for $n = 1$ is explained. This implementation is of practical interest since it provides accurate frequency information with great time resolution, which is then used to obtain accurate amplitude information as well. Finally, some practical results of the application of this method to synthetic and natural sounds are presented at the end of this section.

3.3.1 FT^n: Fourier Transform using n Signal Derivatives

Position, speed, but also acceleration are key parameters in dynamics. There is a deep analogy between these parameters and, respectively, phase ϕ_p, frequency $f_p = \frac{d\phi_p}{dt}$, and also frequency derivative $\frac{df_p}{dt} = \frac{d^2\phi_p}{dt^2}$. In the considered sound model both frequency and amplitude are slow time-varying parameters. More precisely the a_p and f_p functions are band-limited in frequency, with a small frequency much less than the value of smallest f_p. So during a single analysis window of the short-time Fourier transform $\frac{df_p}{dt}$ and $\frac{da_p}{dt}$ are close to 0. We show that under such conditions using the first n signal derivatives can improve Fourier analysis precision both in frequency and amplitude. The idea behind this technique is extremely simple: Differentiating a sine gives a sine, with different phase but same frequency. More formally, signal derivatives must be examined.

Continuous Signal Derivatives

Let us consider a single oscillator:

$$o_p(t) = \mathrm{osc}(f_p(t), a_p(t), \phi_p(0)) = a_p(t)\cos(\phi_p(t))$$

and then let us calculate its first derivative:

$$\frac{do_p}{dt}(t) = \frac{d}{dt}(a_p(t)\cos(\phi_p(t))) = a_p(t)\ \frac{d}{dt}(\cos(\phi_p(t))) + \frac{da_p}{dt}(t)\cos(\phi_p(t))$$

Since a_p is slow time-varying $\frac{da_p}{dt} \simeq 0$ can be assumed so that, when the first term of the sum has a significant value[1]:

$$\frac{do_p}{dt}(t) \simeq a_p(t)\ \frac{d}{dt}(\cos(\phi_p(t))) = -a_p(t)\ \frac{d\phi_p}{dt}(t)\sin(\phi_p(t))$$

But $\frac{d\phi_p}{dt}(t) = 2\pi f_p(t)$ (from Equation 3.3), so:

$$\frac{do_p}{dt}(t) = -a_p(t)\ (2\pi f_p(t))\sin(\phi_p(t)) = 2\pi\ a_p(t)\ f_p(t)\cos\left(\phi_p(t) - \frac{\pi}{2}\right) \tag{3.24}$$

More generally, it is easy to verify by induction that:

$$\frac{d^k o_p}{dt^k}(t) = a_p(t)\ (2\pi f_p(t))^k \cos\left(\phi_p(t) - \frac{k\pi}{2}\right) \tag{3.25}$$

Finally, since differentiation is a linear operation and from Equation 3.1, the k-th signal derivative is:

$$\frac{d^k a}{dt^k}(t) = \sum_{p=1}^{P} a_p(t)\ (2\pi f_p(t))^k \cos\left(\phi_p(t) - \frac{k\pi}{2}\right) \tag{3.26}$$

Analysis Precision from Differentiation

Let us denote by FT^k the Fourier Transform of the k-th signal derivative $\frac{d^k a}{dt^k}$ ($k \geq 0$), and by $|\mathrm{FT}^k(f)|$ the value of its amplitude (magnitude) at frequency f.

The intuition behind Equation 3.26 is that there should be a maximum in every $|\mathrm{FT}^i|$ ($i \geq 0$) power spectrum for each partial p. The underlying idea is then to use $|\mathrm{FT}^i|$ power spectra for different values of i in order to determine the exact frequencies of the set of partials. When only the first n derivatives of a are used, let us call this approach n-order derivative Fourier analysis (and since $\frac{d^0 a}{dt} = a$, the classic Fourier transform coincides with the zero-order derivative Fourier transform FT^0).

Determining the Frequency

A consequence of Equation 3.26 is that for each partial p there is a maximum in every $|\mathrm{FT}^i|$ at frequency f_p (assuming $a_p > 0$ and $f_p > 0$), and:

$$f_p = \frac{1}{2\pi}\frac{\left|\mathrm{FT}^{i+1}(f_p)\right|}{\left|\mathrm{FT}^{i}(f_p)\right|} \tag{3.27}$$

Although this equation may seem of poor interest since it requires the knowledge of f_p, its discrete version will turn out to be of great practical interest. The reason is that the sampling of the FT^i spectra in practice leads to an approximated value f_p^0 of the frequency f_p. The exact value can be then recovered from its approximation using the discrete version of Equation 3.27.

It is extremely important to note that the effects of any analysis window are the same on both $\mathrm{FT}^i(f_p)$ and $\mathrm{FT}^{i+1}(f_p)$ as soon as the same analysis window is used to compute these two spectra. These effects are compensated thanks to the division in the preceding equation.

[1]We are interested only in spectral peaks with a significant amplitude, that is beyond a certain minimal threshold. That is why we will not consider the case where the first term of the sum is not significant, that is when the whole sum is not significant.

Correcting the Amplitude

Of course in order to get precise maxima in a $|\mathrm{FT}^i|$ spectrum (sharp spectral peaks), an analysis window w is needed. Section 3.1 pointed out the consequences of windowing on magnitude spectra. Fortunately, the exact amplitude a_p of partial p can easily be recovered from the approximate analyzed value $a_p^0 = |\mathrm{FT}^0(f_p)|$ since:

$$a_p = \frac{a_p^0}{W(0)} \tag{3.28}$$

Adjusting the Phase

Adjusting phase information is also possible as soon as precise frequency values are available, since for a small time variation Δ_t the model equations ensure that $\phi_p(t + \Delta_t) \simeq \phi_p(t) + 2\pi f_p(t)\Delta_t$. Anyway, this consideration is left aside in this chapter since recovering accurate phase information from analysis is not that important for the considered sound model.

3.3.2 DFT^1: Discrete FT^1 – First Order Fourier Transform

We now describe the discrete version of the FT^n technique for $n = 1$ and shows its practical interest for extracting accurate frequency and amplitude.

The FT^n transform requires the first n signal derivatives. We show that even without any analog device providing such information, the discrete version of FT^1 can be implemented. But since the signal derivative is not usually recorded together with the signal, it has to be calculated from the digital signal after the sampling phase. The discrete signal is uniformly sampled with sampling rate F_s. For all figures and practical examples in the remainder of this section, the audio-CD sampling rate $F_s = 44100$ Hz will be used.

Discrete Signal Derivatives

The first derivative $\frac{da}{dt}$, also noted a', is mathematically defined by:

$$a'(t) = \lim_{\epsilon \to 0} \frac{a(t+\epsilon) - a(t)}{\epsilon}$$

In fact, left and right derivatives for, respectively, ϵ negative or positive are:

$$\begin{aligned} a'_-(t) &= \lim_{\epsilon \to 0_-} \frac{a(t+\epsilon) - a(t)}{\epsilon} = \lim_{\eta \to 0_+} \frac{a(t) - a(t-\eta)}{\eta} \quad (\eta = -\epsilon) \\ a'_+(t) &= \lim_{\epsilon \to 0_+} \frac{a(t+\epsilon) - a(t)}{\epsilon} \end{aligned}$$

Since signal a is a $\mathcal{C}^\infty$ function, left and right derivatives are equal, i.e.:

$$a'_-(t) = a'_+(t) = a'(t)$$

In practice, the audio signal a is discrete, $a[i]$ representing $a(i\ T_s)$, and the smallest (non-zero) possible $|\epsilon|$ for the derivative computation is the sampling period $T_s = 1/F_s$. So let us consider the following approximations:

$$a'_-[i] = (a[i] - a[i-1])\ F_s \quad \text{and} \quad a'_+[i] = (a[i+1] - a[i])\ F_s$$

This time $a'_+[i] \neq a'_-[i]$, but:

$$a'_+[i] = (a[i+1] - a[i])\ F_s = a'_-[i+1]$$
$$a'_-[i] = (a[i] - a[i-1])\ F_s = a'_+[i-1]$$

so apart from a one-sample time translation they are the same discrete functions (and this small translation has a negligible impact on the power spectrum of the signal derivative, which is the only thing considered by the method). Let us arbitrarily take left derivative approximation as an approximation for the derivative itself (taking $a[i] = 0$ for $i < 0$):

$$a'[i] = F_s\ (a[i] - a[i-1]) \tag{3.29}$$

Equation 3.29 has a form which is quite classic in digital filters theory [Moo90, Smi85]; specifically it defines a linear phase high-pass filter. Its equation is:

$$y[n] = F_s\ x[n] - F_s\ x[n-1] \tag{3.30}$$

its transfer function can be easily calculated:

$$H(z) = F_s\ (1 - z^{-1}) \tag{3.31}$$

and finally its gain is:

$$|H(e^{j\omega})| = F_s\ \sqrt{2(1-\cos(\omega))} = 2F_s \sin\left(\frac{\omega}{2}\right) \quad \text{with} \quad \omega = \frac{2\pi f}{F_s} \tag{3.32}$$

Differentiation is a linear operation that can also be considered as a filtering operation with a $2\pi f$ gain (see Equation 3.24), that is $F_s\omega$, which is a theoretical gain quite different from the practical one $|H(e^{j\omega})|$, especially for great values of ω. Figure 3.16 illustrates this phenomenon. Fortunately this difference can be corrected by multiplying the power spectrum of the signal "derivative" approximation obtained from Equation 3.29 by the F scaling factor defined in Equation 3.33.

$$F(\omega) = \frac{\omega}{2\sin\left(\frac{\omega}{2}\right)} \quad \text{with} \quad \omega = \frac{2\pi f}{F_s} \tag{3.33}$$

It is easy to show also that sharp spectral peaks are roughly conserved by a slowly growing gain. Figure 3.17 illustrates this phenomenon. Another possibility is to recover the frequency directly from the gain of the filter. For a given spectral peak, if the v_0 and v_1 (strictly positive) values are measured in the magnitude spectra of, respectively, the signal and its first derivative at the same location corresponding to frequency f, then

$$v_1 = 2F_s \sin\left(\frac{\omega}{2}\right) \cdot v_0$$

thus

$$\omega = 2\arcsin\left(\frac{v_1}{2F_s v_0}\right)$$

that is:

$$f = \frac{1}{\pi F_s} \arcsin\left(\frac{v_1}{2F_s v_0}\right) \tag{3.34}$$

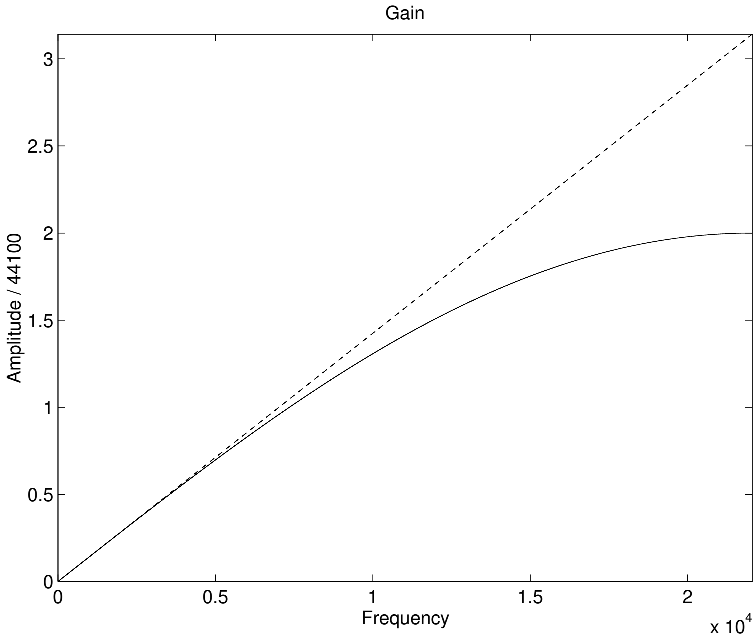


Figure 3.16: Practical gain $|H|$ (solid) versus theoretical differentiation gain (dashed).

In this section the derivative is approximated with a first-order linear filter. In fact, the method could be generalized and many other filters could be used instead since the change in the amplitude of the signal at a certain frequency could be compared with the expected change produced by the filter. The main advantage of using the first difference operator is to remain close to the theory of the continuous method while leading to good results in practice as shown later. If in the future the derivative of the signal could be recorded together with the signal itself, one could just skip the computation of the approximation of the derivative.

Analysis Precision from Discrete Differentiation

Let us denote by DFT^k the Discrete Fourier Transform of the k-th signal derivative ($k \geq 0$), computed using N consecutive samples from a certain location l. $|\mathrm{DFT}^k[m]|$ is the amplitude (magnitude) of bin number m. More formally:

$$|\mathrm{DFT}^k[m]| = \left| \sum_{n=0}^{N-1} \frac{d^k a}{dt^k}[l+n] \; w[n] \; e^{-j\frac{2\pi}{N}nm} \right|$$

For a frequency f_p, the classic Fourier analysis gives the following approximate frequency and amplitude:

$$f_p^0 = m_p \frac{F_s}{N} \tag{3.35}$$

$$a_p^0 = \left|\mathrm{DFT}^0[m_p]\right| \tag{3.36}$$

where m_p is the index of the maximum in $|\mathrm{DFT}^0|$ corresponding to frequency f_p. Taking advantage of $|\mathrm{DFT}^1|$, much more accurate frequency and amplitude values can be obtained.

A consequence of Equations 3.1 and 3.24 is that for each partial p there is a maximum in both $|\mathrm{DFT}^0|$ and $|\mathrm{DFT}^1|$ amplitude spectra for a certain index m_p, as shown in Figure 3.18.

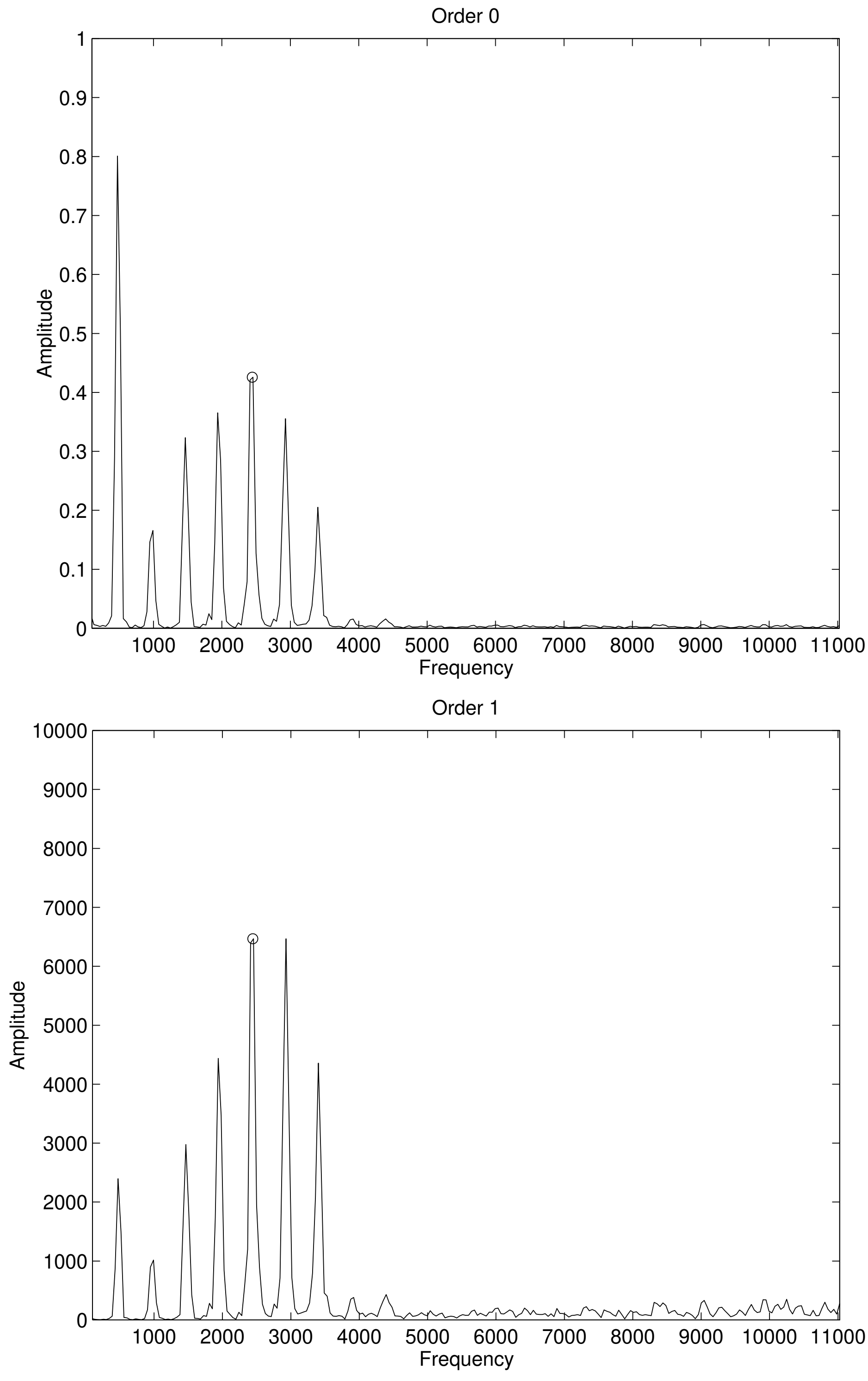
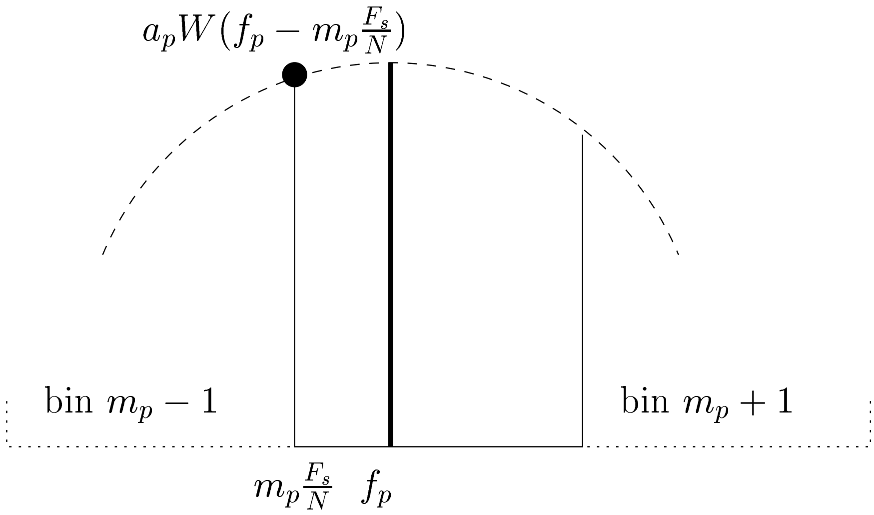
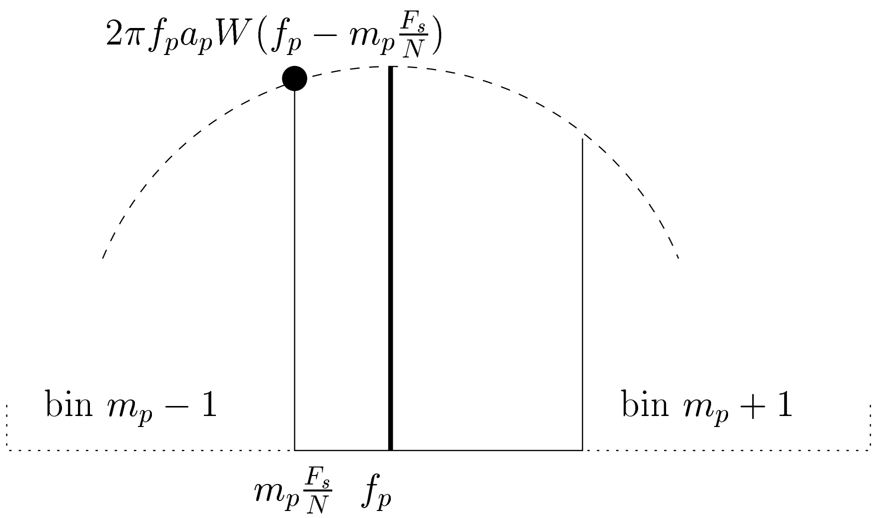


Figure 3.17: DFT^0 and DFT^1 power spectra.



(a) Signal



(b) Derivative

Figure 3.18: Although the analyzed frequency f_p lies in the middle of the m_p-th Fourier transform bin, it produces a peak in the spectra of both the signal (a) and its derivative (b), except that the corresponding amplitudes differ.

Accurate Frequency
A discrete equivalent of Equation 3.27 for $i = 0$ is:

$$f_p = \frac{1}{2\pi} \frac{\left|\mathrm{DFT}^1[m_p]\right|}{\left|\mathrm{DFT}^0[m_p]\right|} \tag{3.37}$$

Precisely, m_p is the closest integer to $f_p \, \frac{N}{F_s}$, $m_p = \lfloor f_p \, \frac{N}{F_s} + \frac{1}{2} \rfloor$ and $\mathcal{B}\,(m_p - \frac{1}{2}) \leq f_p < \mathcal{B}\,(m_p + \frac{1}{2})$. If this is not the case for the calculated f_p, then the DFT^1 analysis has failed for this frequency and this is a clue to bin contamination (there is not a single frequency in this bin).

The exact frequency of every partial can be determined by considering Equation 3.37 for every maximum m in $|\mathrm{DFT}^0|$. Since only the first derivative of a is used, let us call this method the first order derivative Fourier analysis.

Again, as in the continuous case, it is extremely important to note that the effects of any analysis window are the same on both $|\mathrm{DFT}^0[m_p]|$ and $|\mathrm{DFT}^1[m_p]|$ as soon as the same analysis window is used to compute the two spectra. These effects are still compensating thanks to the division in the preceding equation.

Accurate Amplitude
The exact amplitude of each partial can then be determined, from the approximate amplitude a_p^0 and frequency f_p^0 together with the accurate frequency f_p computed before, using the discrete version of Equation 3.28:

$$a_p = \frac{a_p^0}{W\left(\left|f_p - f_p^0\right|\right)} \tag{3.38}$$

It also requires the knowledge of the (continuous) power spectrum W of the analysis window such as those shown in Figure 3.5.

Enhanced "Phase Vocoder" Algorithm

The most famous analysis method for additive synthesis is probably the phase vocoder. Very good introductory texts on the phase vocoder can be found for example in [Moo78c], [Por80], [Por76], [Dol86] or [Ser97a]. It is mainly an implementation of the short-time Fourier transform [All77, AR77]. As a consequence, the classic phase vocoder inherits its limitations from the classic short-time Fourier transform.

The DFT^1 method can be practically used during the analysis stage of spectral modeling synthesis [Mar98], instead of a classic phase vocoder.

An important problem arises with order 2 analysis, due to computation imprecisions during the calculation of the second signal derivative. In fact, for order 1 with 44100 Hz sampling rate, 16-bit samples are just precise enough, since the imprecision resulting from 16-bit quantization (2/65536) is multiplied by $F_s = 44100$ Hz, and then becomes significant. A 24-bit quantization should be a solution. Experiments with high-quality sampling are in progress. For now, only the DFT^1 method has been successfully implemented.

Here is an informal description of a standard analysis algorithm using the DFT^1 method. It is given as a starting point for a practical implementation. For each frame of the temporal signal the following operations have to be done in sequence:

1. Apply windowing function to a;

2. Perform DFT^0 (discrete Fourier transform of this current frame of a);
3. Compute a' using Equation 3.29;
4. Apply (same) windowing function to a';
5. Perform DFT^1 (discrete Fourier transform of this current frame of a');
6. Scale DFT^1 power spectrum by correcting factor F (see Equation 3.33);
7. For each m index of a local maximum in DFT^0 do:
   (a) Compute accurate frequency using Equation 3.37;
   (b) Compute accurate amplitude using Equation 3.38;
   (c) Add pair (frequency, amplitude) to the result list for the current frame.

Since a' can be incrementally computed for each frame and since $|\mathrm{DFT}^1|$ values are needed only for the local maxima of $|\mathrm{DFT}^0|$, two immediate optimizations can be performed, thus saving both space and time. Moreover if the analysis window size is a power of 2, using the Fast Fourier Transform (FFT) to implement the discrete Fourier transform (DFT) makes this algorithm even faster.

Of course the partials must then be tracked from frame to frame in order to recover which (frequency, amplitude) pair belongs to which partial p (see Section 3.4).

3.3.3 Performance and Comparative Results

We now present some results obtained with a direct implementation of the DFT^1 method using the optimized version of the algorithm described above (part of our *InSpect* software package, see Section 3.6), and a very simple peak-tracking strategy (nearest-frequency connection, see Section 3.4).

From the complexity point of view, this method is very interesting since it requires the computation of two small discrete Fourier transforms instead of one much larger. Using the Fast Fourier Transform (FFT), its complexity is the same as for the classic Fourier analysis, that is $O(N\log_2(N))$. But in practice $N = 256$ is often sufficient with the DFT^1 method, whereas the classic Fourier analysis requires much larger values of N (about 8 times larger in practice).

This technique has been successfully tested on both synthetic and natural sounds with low noise. The method can also be generalized for noisy sounds, which vary quickly over time, assuming that the stochastic component remains low. To do so, there is two possibilities: Either revert to order 0, or move to order 2 if computation precision makes it possible. In the first case, the spectral envelope of the noise can be easily found using the classic Fourier transform. In the second case, performing order 2 analysis might help refine the order 1 results to even better isolate spectral peaks.

The DFT^1 method has proven to be very accurate in practice. First, its performance is measured on synthetic examples. A simple comparison shows that the DFT^1 analysis can accurately recover partial parameters where the classic Fourier transform has failed. The same comparison is then shown on natural sounds, including well-known difficult ones like voices with deep vibrato.

Performance on Synthetic Examples

The performance of the DFT^1 method has been measured on synthetic but significant examples consisting of a single oscillator. A single oscillator is in fact a reference example for analysis. Most natural sounds are indeed made of a sum of partials, and since the analysis process is a linear operation, it should also precisely recover the time evolutions of their parameters.

We have compared the DFT^1 method with three of the most accurate analysis methods presented in Section 3.2: the parabolic (quadratic) interpolation using the Brent method, the triangular algorithm, and the spectrum reassignment using analytically-differentiated windows. The full numerical results are available in the appendix at the end of this document.

For most analysis windows, the DFT^1 method has proven to be more accurate than the Brent method. The behavior of the DFT^1 method in the presence of noise is very good. With vibrato or tremolo the results are still satisfactory. The DFT^1 method performs particularly well using the Hann window. Of course a truncated Gaussian window favors the Brent method since the shape of its magnitude spectrum is a parabola, well-suited for parabolic interpolation. With such a truncated Gaussian window the DFT^1 and Brent methods give similar results, excellent ones because the examples we consider have only one frequency component. But in practice a truncated Gaussian window should not be used because of its too large main-lobe width.

We have also compared the DFT^1 method and the accurate triangular algorithm. It turns out that the DFT^1 method has a higher precision in frequency, but lower in amplitude, except for small signal-to-noise ratios (SNR). This might seem surprising, since in our method we use a more accurate frequency information to compute an amplitude value that turns out to be less accurate. In fact the amplitude imprecision might be due to the distortion of the main lobe when the parameters change, as described in the Phase Distortion Analysis in Section 3.2. The triangular algorithm is probably less sensitive to the distortion of the top of the main lobe. Tables A.6 and A.3 (see Appendix) show the comparison between the triangular algorithm and DFT^1 with or without noise.

The spectrum reassignment using analytically-differentiated windows shows excellent results on frequency, equivalent to DFT^1 though, but twice better in the presence of deep vibrato. However the DFT^1 method performs better for noisy sounds with very low SNRs.

We now study the behavior of the DFT^1 method on synthetic examples, with numerical results. The parabolic interpolation using the Brent method is used as a reference.

Base Case: Sinusoidal Oscillator

The first synthetic example considered here is a single oscillator whose frequency is linearly increasing from 440 Hz to 1660 Hz while its amplitude remains constant at 0.8 (see Figure 3.6).

Even with a very small analysis window width of 256 points (less than 6 ms analysis time), the evolutions of the partial shown in Figure 3.6 are almost perfectly recovered. Such a result would have been impossible to achieve with the classic Fourier analysis since a large analysis window is needed to have such a great frequency precision, in which case the time resolution is so bad that the evolution of the frequency with time cannot be successfully recovered. It is important to note that the effect of windowing on the amplitude has been almost completely canceled as well, thanks to Equation 3.38.

Tables A.1 and A.2 show the bias and standard deviation of the errors on frequency and amplitude measured by the DFT (without peak interpolation), the DFT plus parabolic peak interpolation using the Brent method, and the DFT^1 method.

For all these methods the frequency error is inversely proportional to the window width. Since there is only one frequency component, the frequency error of the DFT is independent of the analysis window type. The amplitude precision does not depend on the analysis window width. The DFT^1 method shows excellent results on the amplitude and very good ones on the frequency, depending on the analysis window type. The best relative results are obtained with the Hann window, while a truncated Gaussian window favors the Brent's parabolic (quadratic) interpolation (since the shape of the magnitude spectrum of a Gaussian window is exactly a parabola). With such a window the frequency error is small, even if some big – but very rare – mistakes may lead to a higher standard deviation for the DFT^1 method.

Noise

Let us now add some white noise to the synthetic example considered before. Tables A.4 and A.5 show the errors on frequency and amplitude of the DFT^1 and the Brent methods in the presence of noise and Figure 3.19 illustrates their general behaviors.

The frequency error remains low until the noise amplitude gets close to the amplitude of the signal itself, then the Brent method diverges – because the wrong spectral peak is taken into account – while the DFT^1 frequency error remains proportional to the noise level. The amplitude error remains low until the noise amplitude gets close to the amplitude of the signal itself, then the error increases very quickly for the Brent method while the DFT^1 amplitude error remains proportional to the noise level.

Vibrato and Tremolo

The assumptions about stationarity of the signal are being violated by any vibrato or tremolo on the signal.

The behavior of the DFT^1 method in presence of vibrato is studied using a single sinusoidal oscillator whose frequency is 2000 Hz modulated by a vibrato while its amplitude remains constant at 1. Tables A.7 and A.9 show the errors on frequency and amplitude of the DFT^1 and the Brent methods – using the Hann window – for several vibrato rates and depths. On these tables a boundary indicates when the error is greater than 0.1%. Regarding the DFT^1 method in comparison to the Brent method, this area is wider for the frequency error but smaller for the amplitude error.

For the frequency, the DFT^1 method gives better results than Brent's one when the vibrato rate and depth remain low and worse results as they increase. For the amplitude, the DFT^1 method gives much better results with low vibrato rates and depths. There is a still a degradation when the vibrato is too strong, but the results of the DFT^1 and Brent methods are then equivalent.

The behavior of the DFT^1 method in presence of tremolo is studied using a single sinusoidal oscillator whose frequency remains constant at 2000 Hz while its amplitude is 0.5 modulated by a tremolo. Tables A.13 and A.11 show the errors on frequency and amplitude of the DFT^1 and the Brent methods for several tremolo rates and depths.

The frequency error of the Brent method is constant. The DFT^1 method gives better results, except when the tremolo rate and depth increase. The DFT^1 method also gives lower amplitude errors, equivalent to the results of the Brent method as the tremolo rate and depth increase. The DFT^1 method shows good results provided that the tremolo rate and depth are reasonable, which is the case for most natural sounds.

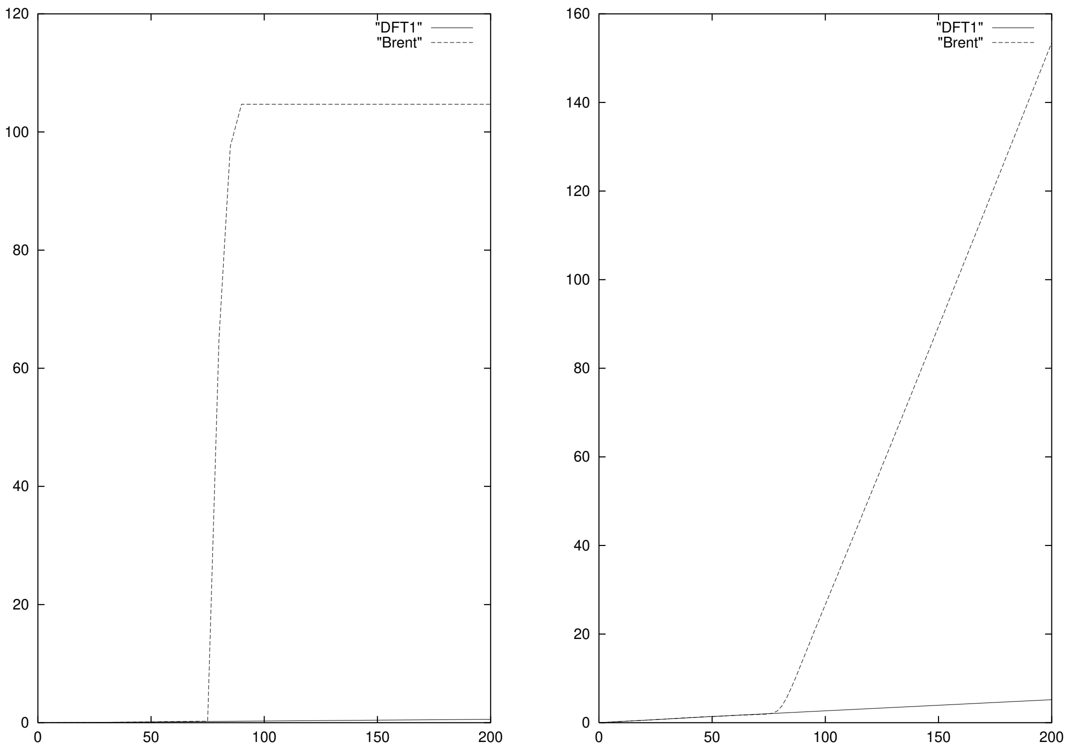


Figure 3.19: Behaviors of the Brent and DFT[1] methods in presence of noise. The errors on frequency (left) and amplitude (right) are displayed as functions of the noise level. The analyzed signal is the same as in Tables A.4 and A.5, and the analysis window is a 512-point truncated Gaussian (favoring the Brent method).

The example in Figure 3.20 shows that the analysis window width should be chosen as small as possible. The only condition is that two frequencies must lie in two separate Fourier transform bins. If the analysis window is too small, this is not the case. And if it is too large, the spectrum is averaged by the Fourier transform and both the classic DFT and the DFT^1 methods perform poorly.

Results on Natural Sounds

The DFT^1 method has also been successfully used to analyze natural sounds with low noise levels. An exhaustive presentation of these results is beyond the scope of this chapter. However, the method succeeds even with polyphonic sounds (see [Cas82]) and voices with deep vibrato. It is well-known that sounds with vibrato are hard to analyze with the classic short-time Fourier transform described in Section 3.1. The reason of this difficulty is illustrated in Figure 3.20.

The examples which are represented in Figure 3.21 are the results of the DFT^1 method in comparison to the classic short-time Fourier analysis on a voice of a soprano singer. The hop size is 64 samples. The Hann analysis window has been used. The best result with the classic Fourier analysis was obtained for an analysis window of 4096 points (93 ms analysis time). This result is at the top of Figure 3.21. With the DFT^1 method, only 1024 points (23 ms analysis time) were needed for the analysis window, and a great quality improvement was achieved. This result is represented at the bottom of Figure 3.21.

Of course the DFT^1 analysis succeeds with classic instruments like guitars, pianos, trumpets, etc. Samples are available on the Internet [Mar00c]. On most of high-pitched sounds (more than 180 Hz for the fundamental frequency), excellent results have been achieved with very short analysis windows, down to 256 points with $F_s = 44100$ Hz, i.e. less than 6 ms analysis time.

Originally designed for sounds with slow time-varying partials, this method has turned out to allow precise analysis of musical instruments with quite fast evolutions (even for the attack phase). This is indeed possible because small analysis windows are sufficient for high-pitched sounds.

3.4 Partial Tracking

Accurately identifying frequency components is the first step for a key technique called partial tracking. It consists in following the evolutions of power spectrum maxima in time. This technique is used in famous software packages like AudioSculpt [IRC96], Lemur [FH96], PARSHL [SS87], and SMS [Ser97b].

More precisely, the short-time spectral analysis produces a series of short-term spectra, from which is extracted the information about the partials. Each short-term spectrum gives birth to a spectral frame, set – possibly empty – of (frequency, amplitude) pairs, corresponding to the frequency and amplitude of the spectral peaks.

The partials must then be tracked from frame to frame in order to recover which (frequency, amplitude) pair belongs to which partial p. This is the role of partial-tracking algorithms.

Partial-tracking algorithms generally maintain a list of active trajectories in the frequency-amplitude plane (functions of time), each corresponding to a partial being reconstructed. The algorithms hop from frame to frame and try to follow the trajectories.

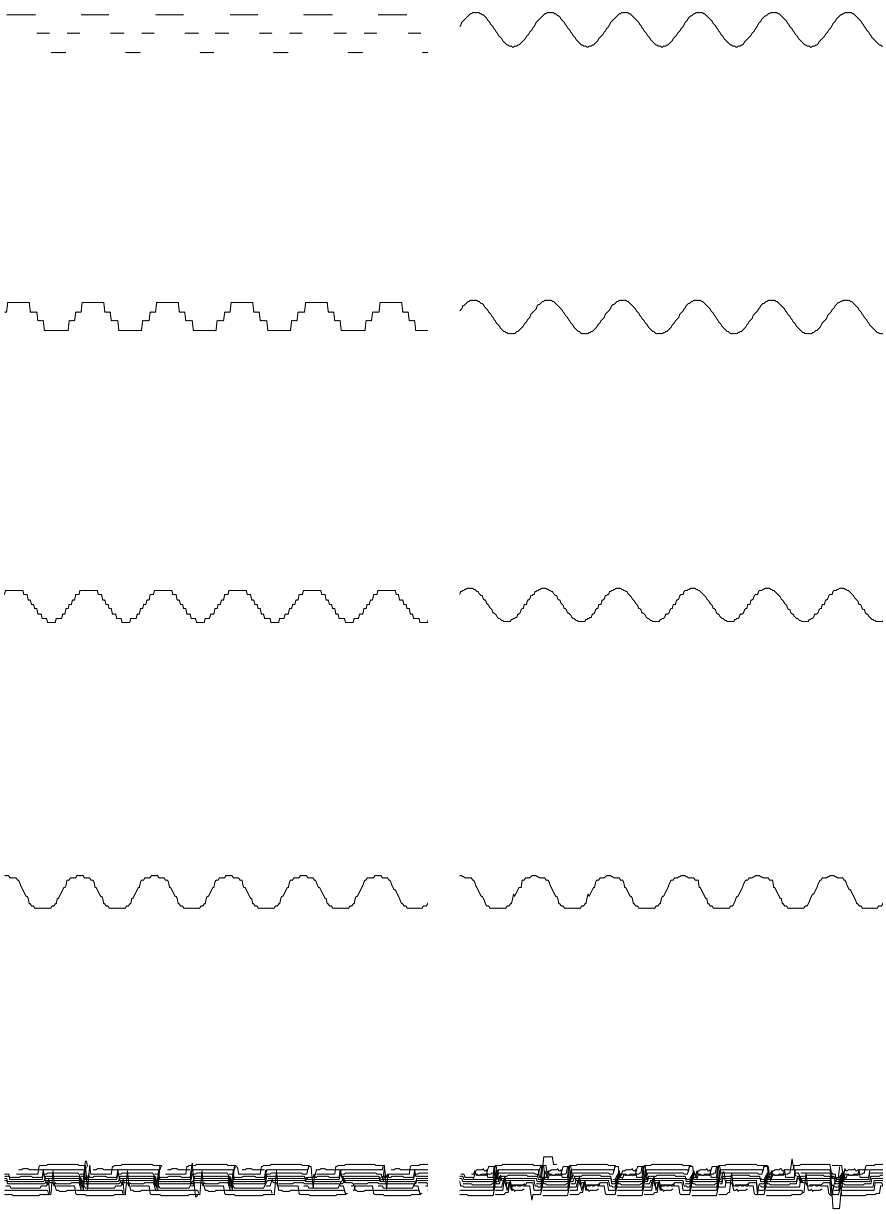


Figure 3.20: Comparison of DFT (left) and DFT[1] (right) on a single partial with vibrato. The analyzed frequency is displayed as a function of time. The analysis window width is increasing from top to bottom (with the 256, 512, 1024, 2048, and 4096 values).

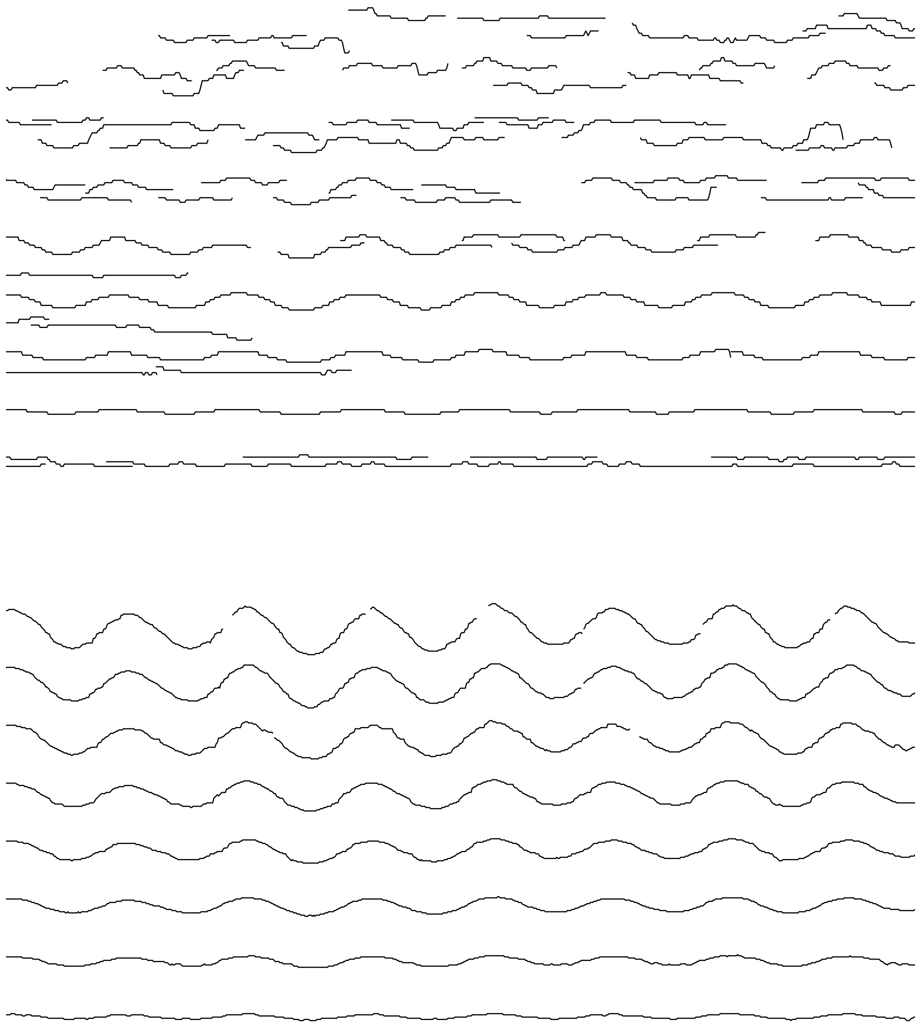


Figure 3.21: Voice with vibrato using the classic DFT (top) and the DFT[1] methods (bottom).

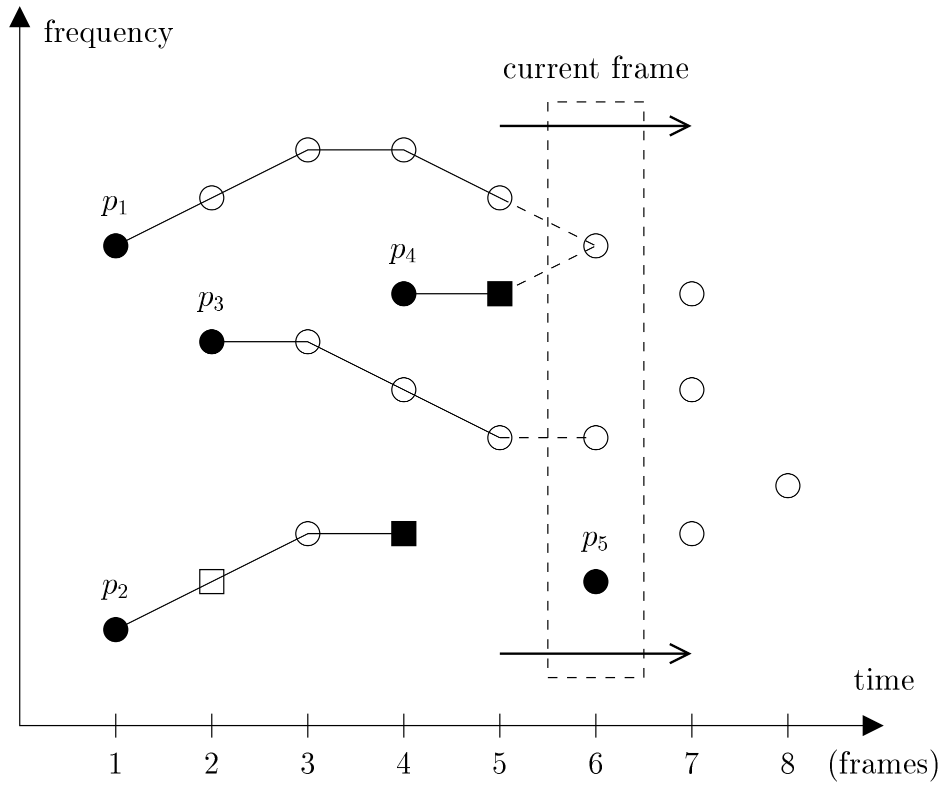


Figure 3.22: Partial trajectories. The circles represent the spectral peaks in the time-frequency plane (the amplitude is not represented). A filled circle or square represent, respectively, the beginning or the end of a trajectory. In the current frame (time 6), the p_3 partial is continued. The p_1 and p_4 partials want the same peak for their respective continuations, but p_1 wins because its connection probability is greater. As a consequence, p_4 is going to die. On the contrary, an unmatched peak gives birth to a new partial, p_5. The p_2 partial is special. At frame 2 no connection was possible, and thus partial p_2 entered the zombie state. Since a connection was possible at the next frame, it then exited from this state and the missing information was reconstructed (empty square).

In a first step (forward search), each trajectory that has not yet found a continuation in the current frame picks out the peak of that frame that matches best, that is the one with the greatest connection probability (see below).

In two-step algorithms the trajectories can choose several peaks (with the same connection probability). Then, in the second step (backward search), all peaks in the current frame choose among the trajectories which have chosen them in the first step, using a certain criterion (for example by minimizing the difference in amplitude).

Finally, the trajectories are extended according to the previous choices and the next frame is considered.

3.4.1 Birth-Death Concept

A trajectory which does not find peak for continuation in the current frame simply "dies". If there are unmatched peaks in the current frame, new trajectories are "born" (see Figure 3.22). The lifetime is the number of frames that a trajectory exists. Of course there is a minimum lifetime to avoid the "blurring" which occurs when analyzing noisy sines without peak tracking for example.

We propose to also add a "zombie state", associated to trajectories which have not found a partner in the current frame but are still active for a few n frames. Those trajectories will die in a few frames, if the connection fails for all these n frames. On the contrary, if such a trajectory finds a peak for continuation in the meanwhile, it exits from its zombie state. Thus the corresponding partial evolutions contain a gap (missing information), which is then filled by an extrapolation technique (such as reconstruction from irregular sampling, see Chapter 2).

This mechanism allows to follow trajectories with small gaps. As a consequence, the short-time analysis can make big mistakes from time to time.

3.4.2 Connection Probability

Each trajectory tries to find a continuation in the current frame by picking out the peak of that frame that matches best, that is the one with the greatest connection probability. The connection probability is a function of the current frequency and amplitude of the trajectory, and of the frequency and amplitude of the candidate for continuation.

The most commonly used connection probabilities are inversely proportional to the distance in frequency between the current frequency of the trajectory and the frequency of the candidate peak, so that the peak with the nearest frequency is chosen for the continuation. This method is called nearest-frequency connection. For the same distance in frequency, we choose to give the priority to the longer (in time) trajectory. Nevertheless, the connection probability can be much more complicated. The amplitude should also be taken into account. Unfortunately it is difficult to design a distance in the frequency-amplitude space. The Euclidean distance is not a good choice because the frequency and amplitude dimensions do not have the same physical unit. The best choice is probably to avoid clicks in the resulting sound, and not to favor the graphical aspect of the trajectories.

Anyway, for each peak in the current frame, the connection probability is computed for the current parameters of each trajectory. The peak with the greatest probability is then considered. If the probability is under a minimal threshold, then no connection is possible and the partial enters the zombie state (see above).

3.4.3 Trajectories Forecast

Each trajectory has a current frequency and amplitude, usually corresponding to the spectral parameters of the last peak chosen for the continuation of the trajectory. If we use these values for the connection probability then we assume that the trajectory is stationary, which is a good approximation for slow time-varying partials. However, it is also possible to try to predict the frequency and amplitude for the next frame using the past evolutions of these parameters.

Apart from using the current frequency and amplitude of the trajectory, the simpler forecast method is probably the linear extrapolation technique. This technique consists in using the frequencies and amplitudes corresponding to the two previous frames, and to consider that the frequency and amplitude corresponding to the current frame should be on the corresponding line. This ensures the continuity of both the trajectory and its first derivative.

Of course more complex extrapolations exist. We can for example use the curvature, to ensure the continuity of the second derivative. Again, the aim is to avoid clicks in the resulting sound, and not to favor the graphical aspect of the trajectories.

Depalle, Garcia, Rodet, and Doval propose in [DGR93, DR93, Gar92] to use statistical modeling, and more precisely the Hidden Markov Models (HMM).

However we prefer to use deterministic methods, for example by using the derivatives of the frequency and amplitude parameters, which can be obtained using the phase distortion analysis (see Section 3.2). Peeters and Rodet successfully use this new technique in SINOLA [PR99] already.

3.5 Extraction of the SAS Parameters

To convert a sound from its temporal representation to the SAS model, one can perform a short-term Fourier analysis, then track partials across short-term spectra, to finally extract the SAS parameters from the set of partials, thus going through three levels of structuring. This structuring is implemented in *InSpect* (see Section 3.6).

After the peak-tracking stage, the result of the analysis is a set of P partials whose parameters (frequency f_p and amplitude a_p) evolve relatively slowly with time. In this section we show how we structure this intermediate representation in order to obtain the parameters of the Structured Additive Synthesis (SAS) model (see Chapter 2).

3.5.1 Frequency and Warping

For harmonic sounds F coincides with the fundamental, possibly missing or “virtual”. For these sounds, if the frequencies were integers, we would have had something like:

$$F(t) = \gcd(f_1(t), \cdots, f_P(t))$$

where *gcd* would be the greatest common divisor. Here is the way to determine the F and W parameters in the general case.

Given a frequency f, let the harmonic frequency distortion for f be:

$$W_f = ((\text{rank}_f(f_1) \cdot f, f_1), \cdots, (\text{rank}_f(f_P) \cdot f, f_P)) \tag{3.39}$$

with $\text{rank}_f(f_i) = [\frac{f_i}{f}]$ ($[x]$ being the nearest integer to x). Denote by F the frequency for which the $y = x$ line is closest to the $(\text{rank}_F(f_i) \cdot F, f_i)$ points (in the least-squared error terms). Then, F is the frequency parameter of the model and the warping W is the continuous version (interpolation) of W_F. This interpolated version can be reconstructed from its samples using the techniques described in Chapter 2.

3.5.2 Amplitude and Color

In the additive representation, the amplitude A corresponds to the sum of the amplitudes of all partials and can be calculated from the additive parameters using Equation 3.40. In order

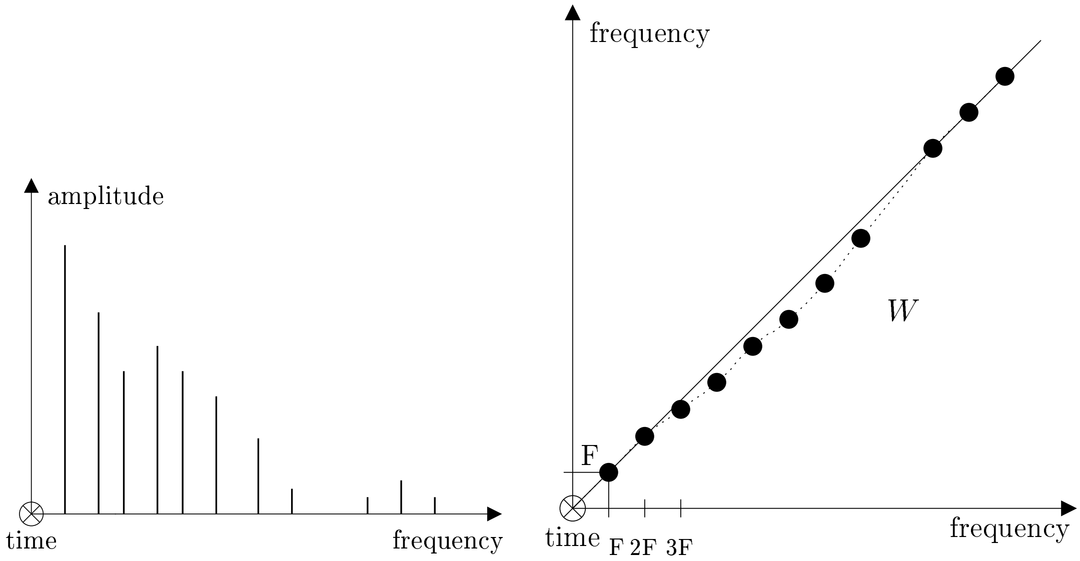


Figure 3.23: The spectrum of a non-harmonic sound and the corresponding frequency F and warping W parameters.

to consider the RMS (Root Mean Square) amplitude (closer to the perception), Equation 3.41 must be used instead of Equation 3.40.

$$A(t) = \sum_{p=1}^{P} a_p(t) \tag{3.40}$$

$$A_{\mathrm{RMS}}(t) = \frac{1}{\sqrt{2}} \sqrt{\sum_{p=1}^{P} (a_p(t))^2} \tag{3.41}$$

Color is simply the continuous version (interpolation) of the spectral envelope defined by the (f_p, a_p) points. In practice, we reconstruct it as a signal using the techniques described in Chapter 2, and then we resample it uniformly with the lowest rate possible.

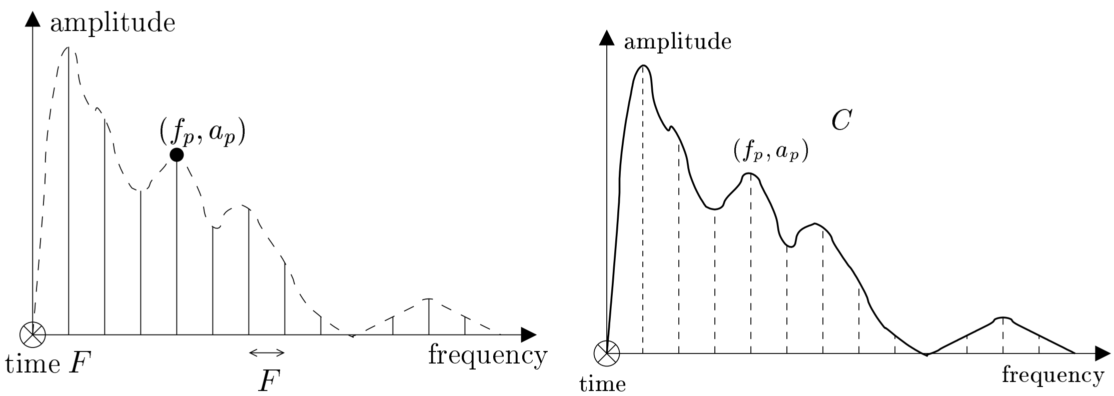


Figure 3.24: An harmonic sound at time t, and its color C.

3.6 *InSpect* Software Package

InSpect ("Inspect Spectrum") [MS99] is a sound analysis program, designed to look at the inner structures of sound. It performs the analysis of sampled sounds, extracting parameters and structuring them into spectral sound models. The resulting spectral sounds can then be manipulated in a very musical and intuitive way. Although *InSpect* can resynthesize these sounds, *ReSpect* was specially designed for the purposes of real-time synthesis (see Chapter 4).

InSpect was initially written in C plus *Tcl/Tk* [Mar00c], then rewritten using *ProSpect* as a starting point (see Chapter 2) [Mar00b]. This program generally allows to go from a sound model to another one, and particularly to obtain the sound in a spectral model from its temporal representation.

Many analysis methods are proposed (see Sections 3.1, 3.2, and 3.3) but few were implemented in a software system, and it appears that freely-available analysis software tools are extremely rare. We have implemented these methods in *InSpect*. Our software system is developed for the Linux operating system (although a version for Windows 95/98/NT also exists), and is distributed freely under the terms of the GNU General Public License (GPL) [FSF91].

InSpect was developed in order to convert sounds from the temporal model (audio signal amplitude versus time) to a spectral representation. Although it was first designed for low-noise sounds, it can also analyze, transform, and resynthesize noises as well. But *InSpect* does not yet separate the sinusoids (deterministic part) from the noise (stochastic part) as SMS [Ser97b] does. However, this functionality will be available in the forthcoming version, using recent techniques such as the one proposed by Peeters and Rodet in [PR98].

InSpect has numerous functionalities that are extensions of *ProSpect* and thus can be reused in other programs. Among these functionalities are many short-time analysis methods (and in particular the n-order Fourier transform, see Section 3.3), partial-tracking techniques, as well as algorithms for extracting musical parameters (pitch, volume, brightness, etc.) from the sounds. There are also many spectral synthesis methods, thus providing a very convenient way to compare their respective performances. We have implemented the fastest synthesis method directly as a module for the kernel of the Linux operating system (see Chapter 4), and *ProSpect* takes advantage of it to perform real-time spectral synthesis.

3.6.1 Overview

Basically, *InSpect* can open a sound file, analyze it, and synthesize a new one from the analysis. The synthesized sound file may be played or exported. *InSpect* does neither care about recording, nor about the way the original sound file was obtained. Neither does it compare the resulting sounds, which is strictly the role of the (human) user. Figure 3.25 gives a general overview of the main functionalities of *InSpect*. *InSpect* allows the user to look at the inner structures of sound. For each partial it is possible to show its frequency and amplitude as functions of time. It is also possible to display the short-time spectrum at a given time and the associated spectral envelope, as well as spectrograms, phasograms, and results of linear prediction.

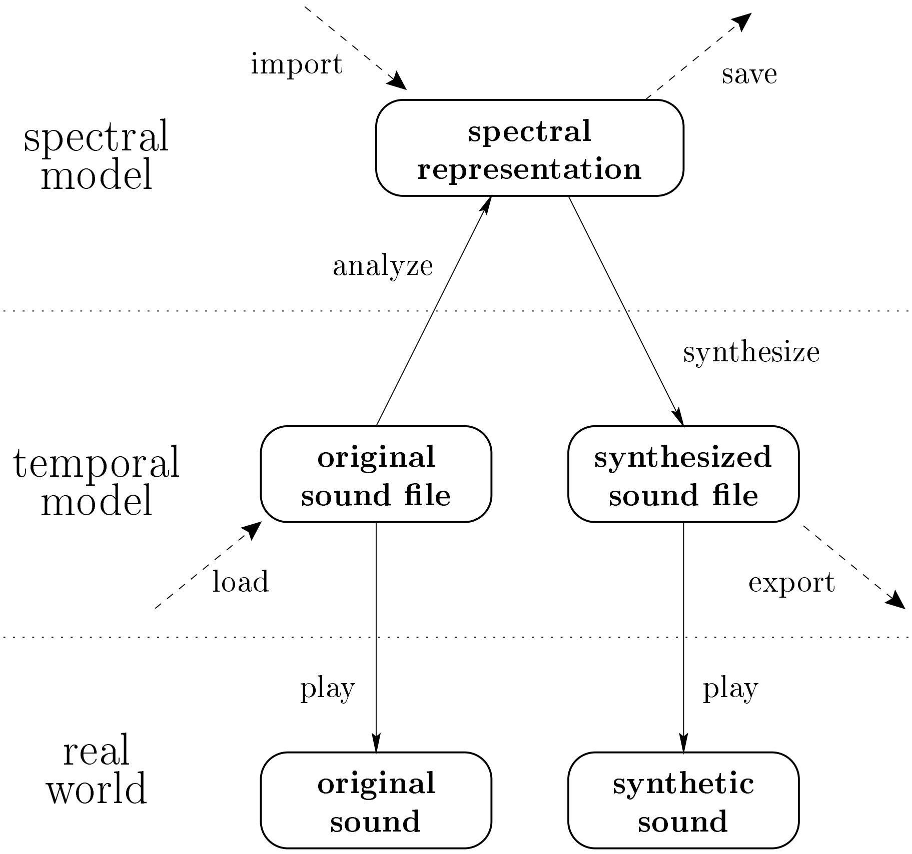


Figure 3.25: *InSpect* architecture overview.

Step 1: Loading

The first step is to load a sampled sound. *InSpect* supports many file formats. Traditional computer sound files (WAV, AIFF, etc.) are typical examples of representations of sounds in the temporal model.

Step 2: Analysis

InSpect was designed not only for researchers and engineers, but also for composers and musicians. That is why the analysis step can be performed without setting lots of parameters. The most useful analysis configurations are available instead, according to the nature and the pitch of the sound to analyze. Moreover the default configuration succeeds in most cases. The default method of *InSpect* works best with low-noise sounds, harmonic or not. The analysis step and its configuration are developed later in this section.

InSpect can also structure these additive parameters in order to switch to the Structured Additive Synthesis model (SAS) (see Section 3.5). But since this functionality is still experimental in the actual version of the software tool, we will not develop it further in this section.

Step 3: Saving

When the analysis step is completed, the resulting spectral sound is displayed as shown in Figure 3.26 and it is possible to save it in a spectral format. *InSpect* knows how to take advantage of the slow time-varying nature of the parameters in order to perform efficient compression. It is also possible to import directly an already-analyzed sound, thus skipping the analysis step.

3.6.2 Analysis

Given a sampled sound, the aim of the analysis stage is to decompose it into elementary sound components called partials, whose frequencies and amplitudes evolve slowly in time. This is the well-known McAulay-Quatieri analysis [MQ86] used in Lemur [FH96] and SMS [Ser97b]. *InSpect* performs this analysis in three steps.

Step 1: Short-Time Analysis

Our analysis tool can use the new DFT[1] technique (see Section 3.3) for extracting parameters from sampled sounds. In addition to the signal itself this new technique uses the derivative of the signal. We compute the discrete Fourier transform (using the FFT algorithm) of both the signal and the derivative of the signal. The quotients between peaks across these two spectra allow us to compute very accurate values for the instantaneous frequencies of the sound, which are then used to precisely compute the corresponding amplitudes. Our method gives very accurate values for frequency and amplitude contents, provided that partials are separated by at least the lowest frequency of the Fourier transforms used. Experience has shown that our method can work with very short analysis windows, speeding up the analysis while maintaining a high-quality result together with an increased time resolution.

InSpect was first developed in order to experiment in practice this enhancement of the classic short-time Fourier analysis called the n-order Fourier analysis [DCM00]. However

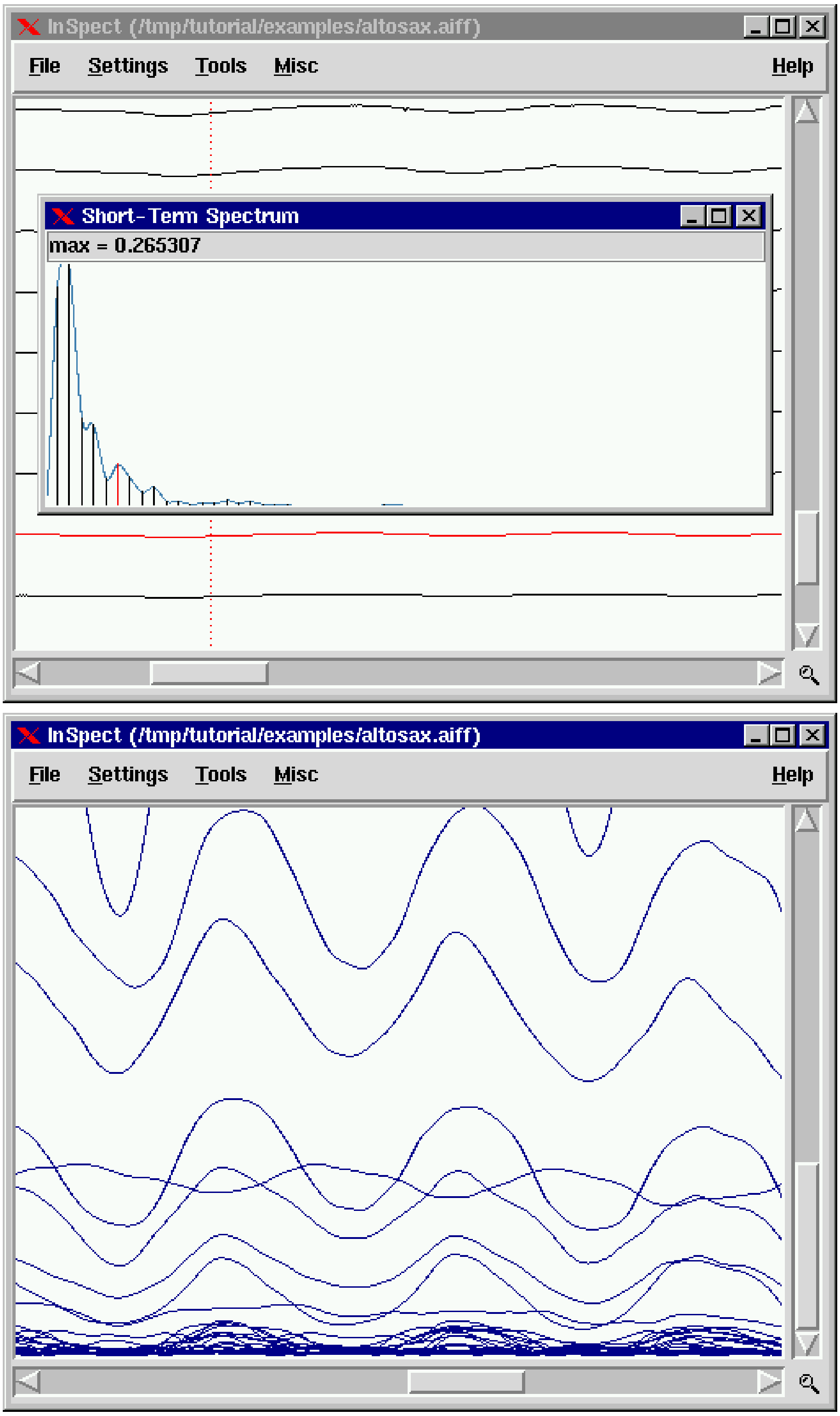


Figure 3.26: *InSpect* displaying the evolutions of the partials of an alto saxophone as functions of time during 0.7 second. The snapshot on the left shows the frequencies of the partials, as well as a short-time spectrum and the corresponding spectral envelope, while the one on the right shows the amplitudes of the partials.

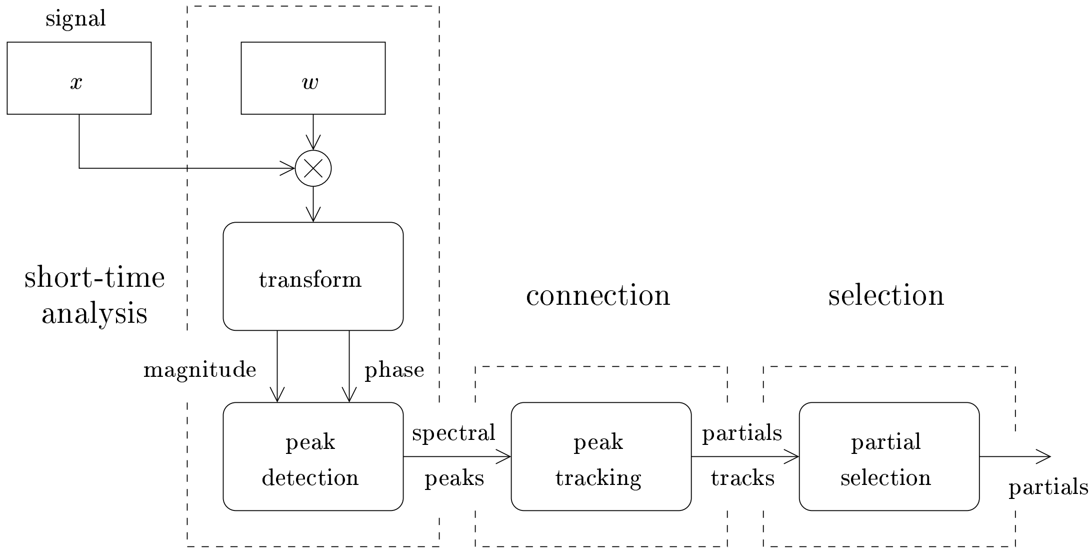


Figure 3.27: Analysis process, consisting of three steps (short-time analysis, partial connection, and partial selection). During the short-time analysis step the signal x is multiplied by the analysis window w, then transformed into a spectral representation. The peaks in the magnitude spectra are tracked from frame to frame to form partials during the connection step. Finally, the selection step keeps the partials according to a certain criterion.

other analysis methods are implemented to make comparisons possible, like the one described in Section 3.2. Among the analysis parameters you can change are the analysis window width and type (Hann, Kaiser, Bartlett, Hamming, Blackman, etc.).

Step 2: Connection of Partials (Partial Tracking)

Each short-time analysis produces a spectral frame. Spectral peaks are tracked from frame to frame to form partials. Different connection strategies are available, as well as connection probabilities and thresholds in frequency and amplitude.

Step 3: Selection of Partials

The result of the connection step is a (frequently large) set of partials. *InSpect* selects only some of these partials according to certain selection methods and criteria. The default method is selecting the strongest and longest partials, given thresholds in strength (amplitude) and duration (time). Thus, partials that have a low amplitude level or that are too short are removed from the set of partials. This is very useful when analyzing noisy sounds with a low noise level.

3.6.3 Transformations

The result of the analysis is a set of partials whose parameters (frequency and amplitude) evolve relatively slowly with time. Our software system allows us to structure and manipulate this intermediate representation. We provide musical parameter extraction (volume, pitch, brightness, etc.) as well as traditional operations such as filtering, time-stretching (without limits on the stretching factor), cross-synthesis, transposition while preserving formants, timbre morphing and many more. Since these manipulations are made on a spectral representation which has fewer data points than the signal itself, our operations are quite efficient. These manipulations are in fact done by *ProSpect* (see Chapter 2), on the top of which *InSpect* is now developed.

3.6.4 Resynthesis

From the spectral representation, it is then possible to resynthesize a sound back in the temporal model. By playing the original and resynthesized sounds and comparing them, one can hear if the original sound has been faithfully modeled.

Again different synthesis methods are proposed, mainly for the purposes of research and experimentation. The fastest synthesis method is available in a separate software tool called *ReSpect* (see Chapter 4).

3.7 Compression of Sinusoidal Modeling Parameters

We have implemented in *InSpect* a technique for lossless compression of the sinusoidal modeling parameters [Mar00a]. This technique can be very useful for any analysis / synthesis program dealing with spectral modeling. Compression is also useful for embedding spectral sounds in a synthesizer, broadcasting spectral sounds or simply storing many of them on a medium.

3.7.1 Compression Method

This technique consists in compressing the frequency and amplitude parameters of each partial by adaptively sub-sampling and encoding their evolutions with time. It can be easily extended to handle spectral envelopes instead of partials, that is two-dimensional structures instead of one-dimensional ones.

Spectral models based on additive synthesis contain a deterministic part consisting of a – often huge – number of partials, which are pseudo-sinusoidal tracks for which frequencies and amplitudes evolve slowly with time. The spectral modeling parameters of this deterministic part consist of the evolutions in time of the controls of the partials, thus leading to a large amount of data.

Grey [Gre75] demonstrated that for many natural sounds the functions f_p and a_p could be simplified by piecewise-linear approximations with great data reduction by keeping only some breakpoints instead of every discrete value of the (sampled) functions. In fact, even if the consequences of this simplification are often not noticeable, they can become very audible after some transformations have been performed on the sound. Strawn [Str80] also proposed to approximate the frequency and amplitude functions of the partials. Charbonneau propose in [Cha80] a study of the perceptual effects of these data reductions on the timbre of the sounds. The compression scheme we consider in this section is a lossless compression. We consider the frequency and amplitude functions of the partials as continuous-time control signals.

Sub-Sampling

The frequencies and amplitudes of the partials are slow time-varying parameters, slow enough to avoid amplitude or frequency modulation phenomena that would modify the timbre. They can be regarded as inaudible signals controlling audible oscillations. They are indeed band-limited signals, with a maximal frequency F below the threshold of hearing, around 20 Hz. The Shannon-Nyquist theorem assures that $2F$ – only 40 here – samples per second are sufficient for the signal to be reconstructed without any error. The evolutions in time of the frequency and amplitude of all the partials can be safely sub-sampled. For that purpose we use a generic resampling algorithm such as the one proposed by Smith in [SG84, Smi00].

Encoding

After the resampling of the parameters, an encoding process is performed in three steps on each parameter (frequency and amplitude) of every partial of the sound.

Delta-Encoding. First, delta encoding consists in replacing the stream of parameter samples by the one resulting from the differences between two consecutive samples, close samples leading to small differences:

$$v_1, v_2, \cdots, v_n \rightarrow v_1, (v_2 - v_1), \cdots, (v_n - v_{n-1}) \tag{3.42}$$

Variable-Length Quantities (VLQ). Then, variable-length quantities – also used in MIDI files [MMA96] – allow us to store each delta-encoded value with a reduced number of bits. Each value is stored as a series of bytes which is called a variable-length quantity. Only the first 7 bits of each byte are significant. So, if we have a value requiring more than 7 bits, we have to unpack it into a series of 7-bit bytes. Of course, we will have a variable number of bytes depending upon our value. To indicate which is the last byte of the series, we leave bit 7 clear. In all of the preceding bytes, we set bit 7. So, if the value is between 0 and 127, it can be represented as one byte. More generally, if the value is between 2^{7k} and $2^{7(k+1)} - 1$, $k + 1$ bytes are required for its corresponding variable-length quantity.

Run-Length Encoding (RLE). Finally, a classic run-length encoding algorithm is used to compress the sequences of consecutive samples with the same value. Run-length encoding stands for the specification of elements in a list as a list of pairs giving the element and number of times it occurs in a run. This is a well-known technique often used in mathematics and computer science. Here is an example of run-length encoding:

$$1, 1, 1, 2, 3, 3, 4, 4, 4, 4, 4 \rightarrow (1, 3), (2, 1), (3, 2), (4, 5)$$

This technique allows to compress the list of values whenever consecutive values are the same.

3.7.2 Compression Enhancement

It is possible to enhance this basic compression scheme.

size (in Kbytes)	sine	voice	saxophone	trumpet
original	86	127	246	3190
spectral	11	1077	1137	13134
resampled	<1	270	284	3284
Delta	<1	135	142	1642
VLQ	<1	30	30	660
RLE	<1	12	15	589
spectral (zip)	9	753	1051	10449

Table 3.7: Some results using the compression method. The last line gives the performance of the Lempel-Ziv method (zip files).

Adaptive Sub-Sampling

First, it is possible to enhance the compression ratio by performing an adaptive sub-sampling, that is by choosing the minimal sampling rate allowed for the parameters f_p and a_p of each partial p instead of fixing an unique sampling rate for all the partials during the resampling step.

Interdependent Parameters

It is also possible to enhance the compression ratio by taking advantage of the potential interdependence of the parameters.

Pseudo-Harmonic Sounds. For pseudo-harmonic sounds like in Figure 3.28(a) we use the knowledge that the frequencies are close to multiples of the fundamental frequency. We store only the ratios to this frequency, which are close to integers.

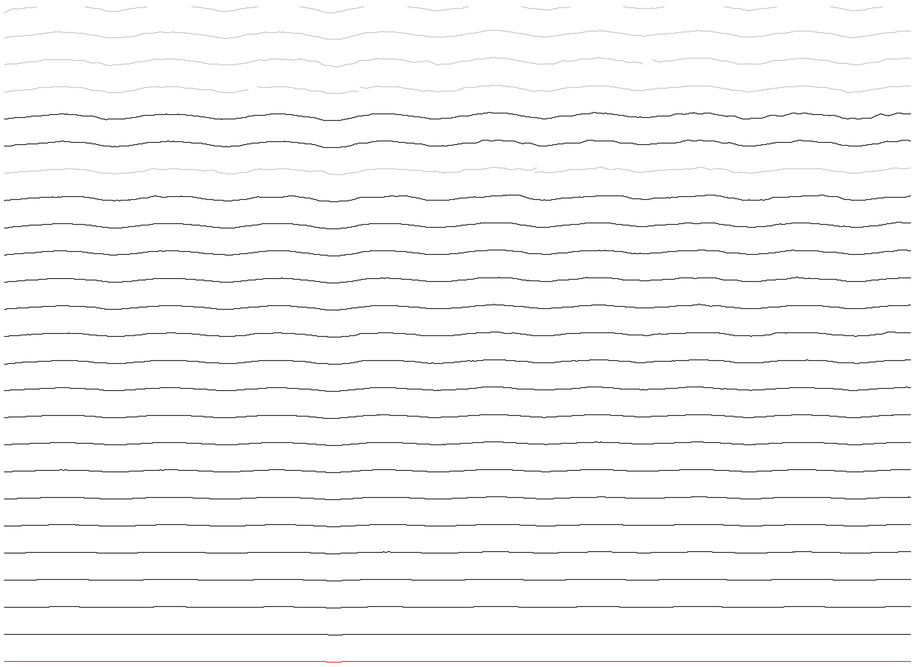
Analysis of Parameters. Such relations among partials can also exist for their amplitudes, as in Figure 3.28(b), and can be found out by analyzing the parameters themselves and comparing the results among partials (see Section 3.8).

3.7.3 Results

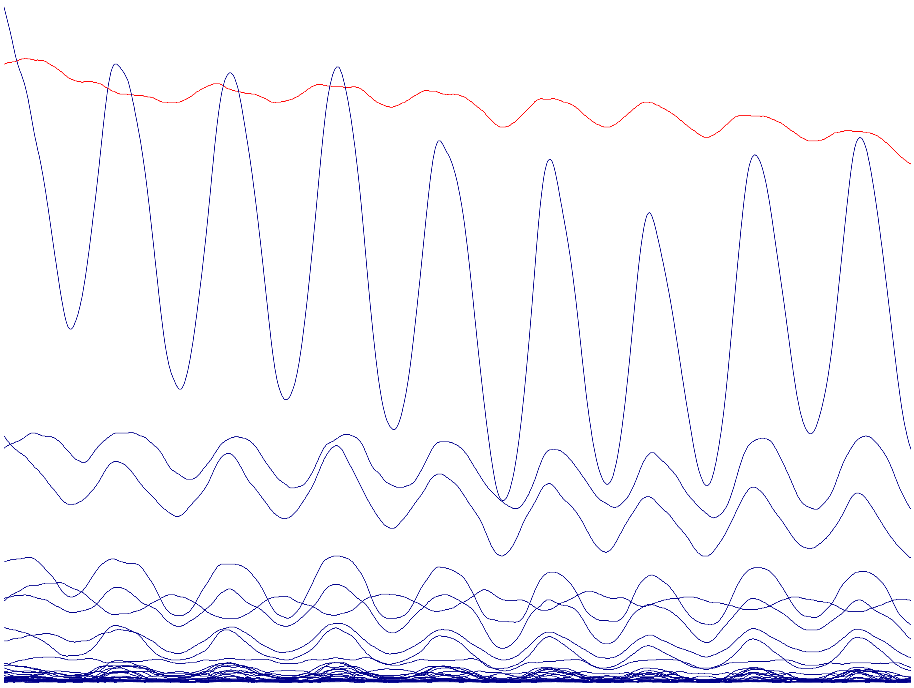
Concerning the compression ratio, the saxophone sound partly shown in Figure 3.28 lasts 2.85 seconds and was sampled at 44100 Hz. When the spectral analysis is done each 64 samples, this sound gives rise to an SDIF file of 1137 Kbytes (using sinusoidal tracks with 64-bit floating point values). After the resampling and delta-encoding steps, its size is only 284 Kbytes. After the VLQ step, this size drops to 30 Kbytes. At the end, after the RLE step, it is only 15 Kbytes. In this case there is a 1/75 ratio from the SDIF information. Other examples can be found in Table 3.7. Moreover, we show that the well-known Lempel-Ziv method – used for example in the zip compression – performs poorly on the spectral data.

3.7.4 File Format

The resulting structures are not suited for implementation in the Sound Description Interchange Format (SDIF) [WCF+99, CNM00, IRC00], which is overall a succession of frames



(a) Frequencies



(b) Amplitudes

Figure 3.28: The evolutions of the partials of an alto saxophone during 1 second. The frequencies (a) and amplitudes (b) are displayed as functions of time (horizontal axis).

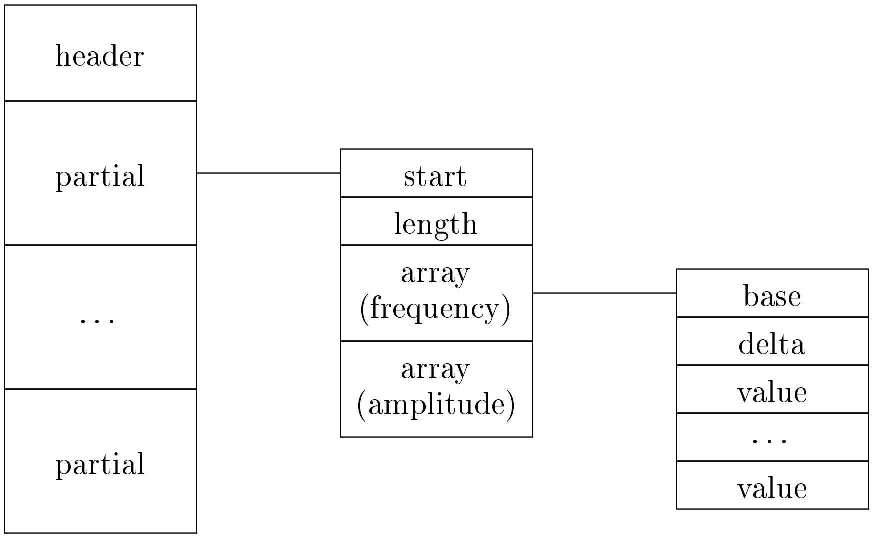


Figure 3.29: The hierarchical structure of the MSC file format.

ordered in time. This is the reason why we have designed a new file format for (compressed) spectral sounds. This format is currently known as the MSC format in the *InSpect* analysis tool [MS99, Mar00c, Mar00b].

The detailed description of the MSC format can be found in the sources of *InSpect*. For short, this format is composed of a header followed by the compressed partials. Each partial consists of two values (start and length) followed by two compressed arrays, one for the frequency and the other for the amplitude of the partial. Each compressed array is composed of the stream of values of the down-sampled evolutions of the considered parameter of the partial. The first sample is called the "base", and the amplitude of the variations is called the "delta". In fact, the values (v_i) are stored relatively to the base and normalized according to the delta, that is (v_i − base)/delta, encoded as integers using variable-length quantities (see above).

We perform lossless compression with impressive ratios and the file format we have designed is very compact. It has proven to be very useful for analysis / synthesis programs based on spectral modeling. Even if we have not investigated it yet, we believe that our technique could be used for lossy but perceptually lossless compression, using quantization and taking advantage of psychoacoustic phenomena such as masking.

3.8 Perspective: Analyzing the Spectral Parameters

In this section, we show how the reanalysis of the parameters coming from initial analysis could turn out to be extremely useful not only to enhance the compression ratio, but also to perform very interesting musical processing on the tremolo or vibrato of the sounds for example. This reanalysis provides us with further research topics since it turns out to be of great interest for pitch tracking and source separation for example.

3.8.1 Spectrum of a Spectrum

In our FT^n analysis method (see Section 3.3), we take advantage of two Fourier transforms computed in parallel. We show here that the use of two Fourier transforms in sequence is of great interest too.

More precisely, we consider the magnitude spectrum of the Fourier transform of the magnitude spectrum of the Fourier transform of the signal. Let us denote by "Fourier of Fourier transform" this combination of the two Fourier transforms.

Note that this transform is not the same as the well-known "cepstrum", which is the (inverse) Fourier transform of the logarithm of the spectrum resulting from the Fourier transform.

This transform is well-suited for pitch-tracking, that is for computing the fundamental frequency of the sound, even if it is missing or "virtual".

For example, if we consider an harmonic sound, its Fourier transform has a series of peaks in its magnitude spectrum corresponding to the harmonics of the sound, at frequencies close to multiples of the fundamental frequency F. Some harmonics may be missing, even the fundamental itself. Anyway, the Fourier of Fourier transform of an harmonic sound shows a series of peaks, and the first and most prominent one corresponds to the fundamental frequency F of the harmonic sound, and its amplitude is the amplitude A of the sound (more precisely the sum of the amplitudes of the harmonics of the sound). Figure 3.30 illustrates this.

In the spectrum resulting from the first Fourier transform (FT), the index of a bin i_{FT} is related to the analyzed frequency f. More precisely, if F_s is the sampling rate and N the size of the Fourier transform, we have:

$$i_{\text{FT}} = f\,\frac{N}{F_s} \tag{3.43}$$

When considering an harmonic sound whose fundamental frequency is F, the magnitude spectrum shows a series of uniformly-spaced peaks (unless some harmonics are missing). The distance between two consecutive harmonics is F, which corresponds to a period of Δ bins where:

$$\Delta = F\,\frac{N}{F_s} \tag{3.44}$$

In the spectrum resulting from the Fourier transform of the magnitude spectrum of the first Fourier transform (FT(FT)), the greatest local maximum of magnitude (apart from the one corresponding to bin 0) is located at the bin corresponding to index:

$$i_{\text{FT(FT)}} = \frac{1}{\Delta}\,\frac{N}{2} \tag{3.45}$$

In Equation 3.45 we consider that the size of the second Fourier transform is again N. This is no mandatory though. It is then possible to recover the fundamental frequency from the value of this index:

$$F = \frac{F_s/2}{i_{\text{FT(FT)}}} \tag{3.46}$$

The same reasoning also works for single sinusoids or rippled noises (even if some ripples are missing). Figure 3.31 illustrates this. As a consequence, the Fourier of Fourier transform turns out be be extremely well-suited for determining the pitch of the sounds, as well as the volume. We have also verified this for natural sounds, as shown in Figure 3.32. It is important to note that the amplitude corresponding to the $i_{\text{FT(FT)}}$ index is close to the sum of the amplitudes of the harmonics constituting the sound, that is the amplitude A in the SAS model. One can

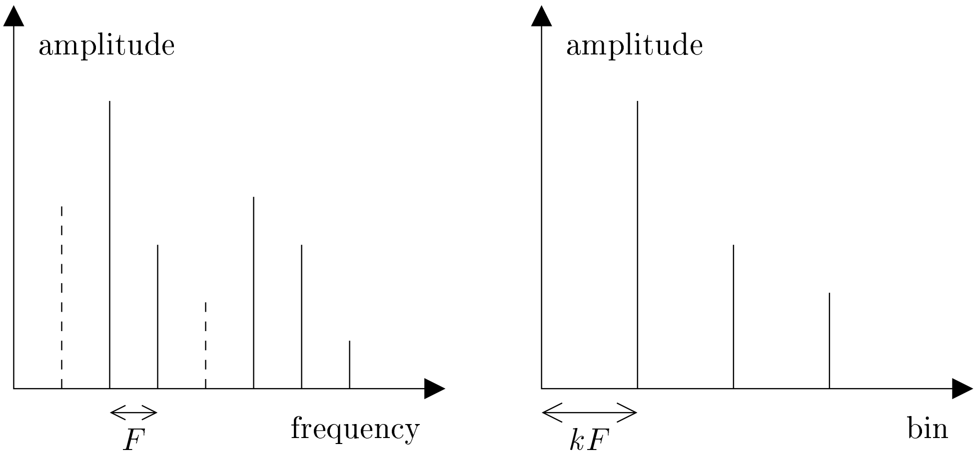


Figure 3.30: The power spectrum of an harmonic sound (left) together with the power spectrum resulting from the Fourier transform of this first spectrum (right). There might be missing harmonics (dashed).

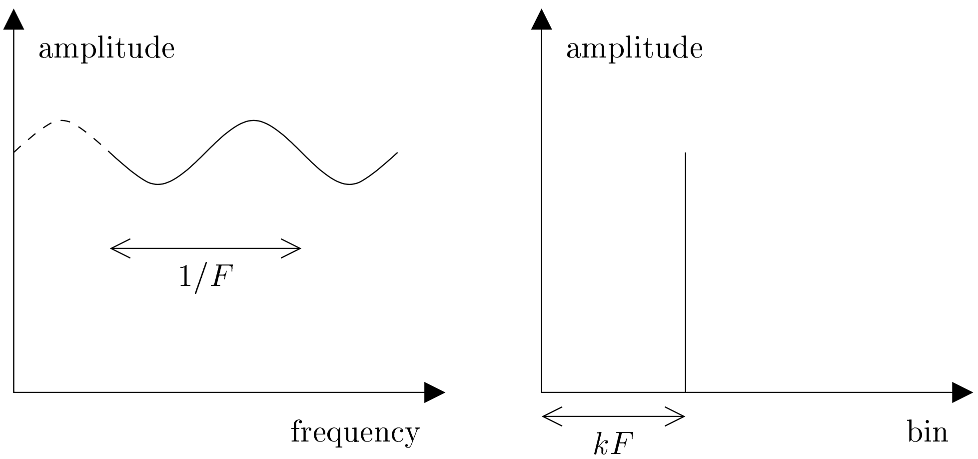


Figure 3.31: The power spectrum of a rippled noise (left) together with the power spectrum resulting from the Fourier transform of this first spectrum (right). There might be missing ripples (dashed).

also obtain instead a good approximation of the RMS amplitude, by replacing the amplitudes by their squares in the magnitude spectrum prior to the second Fourier transform, and by replacing the amplitudes by their square roots in the magnitude spectrum resulting from this second transform.

3.8.2 Analysis of Partials Evolutions

Performing a spectral analysis on the parameters of the partials is of great interest too. More formally, we consider the classic sinusoidal model, where the audio signal a is given by equations:

$$a(t) = \sum_{p=1}^{P} a_p(t) \cos(\phi_p(t)) \tag{3.47}$$

$$\frac{d\phi_p}{dt}(t) = 2\pi f_p(t) \quad \text{i.e.} \quad \phi_p(t) = \phi_p(0) + 2\pi \int_0^t f_p(u)\ du \tag{3.48}$$

where P is the number of partials and the functions f_p, a_p, and ϕ_p are the instantaneous frequency, amplitude, and phase of the p-th partial, respectively. Since the amplitude and frequency of each partial can be considered as band-limited signals, we now decompose them within a similar sinusoidal model. More formally:

$$\left\{ \begin{array}{c} a_p(t) = a_{a_p,0}(t) + \sum_{n=1}^{P_{a_p}} a_{a_p,n}(t) \cos(\phi_{a_p,n}(t)) \\ \frac{d\phi_{a_p,n}}{dt}(t) = 2\pi f_{a_p,n}(t) \quad \text{i.e.} \quad \phi_{a_p,n}(t) = \phi_{a_p,n}(0) + 2\pi \int_0^t f_{a_p,n}(u)\ du \end{array} \right. \tag{3.49}$$

and

$$\left\{ \begin{array}{c} f_p(t) = a_{f_p,0}(t) + \sum_{n=1}^{P_{f_p}} a_{f_p,n}(t) \cos(\phi_{f_p,n}(t)) \\ \frac{d\phi_{f_p,n}}{dt}(t) = 2\pi f_{f_p,n}(t) \quad \text{i.e.} \quad \phi_{f_p,n}(t) = \phi_{f_p,n}(0) + 2\pi \int_0^t f_{f_p,n}(u)\ du \end{array} \right. \tag{3.50}$$

where $a_{a_p,0}$ and $a_{f_p,0}$ are extremely slow time-varying functions, band-limited to a frequency much lower than the smallest $f_{a_p,n}$ and $f_{f_p,n}$ frequencies ($n > 0$), in practice a few Hz. These $a_{a_p,0}$ or $a_{f_p,0}$ parameters define the macroscopic variations in, respectively, amplitude or frequency – that is the "envelopes" – whereas the other parameters (for $n > 0$) reflect the microscopic variations. This decomposition is illustrated in Figures 3.33, 3.34, and 3.35. By performing a spectral analysis, we can extract from each amplitude or frequency parameter of each partial an envelope together with pseudo-partials, $(f_{a_p,n}, a_{a_p,n})$ or $(f_{f_p,n}, a_{f_p,n})$ (for $n > 0$), respectively, that are in fact the evolutions of the spectral parameters resulting from the reanalysis of the parameter of the partial. Thus the analysis of the evolutions of the partials results in other (pseudo) partials, very slow time-varying. In this further analysis level we can find similarities among the parameters. For example, a sound with tremolo or vibrato will show such similarities among the partials. It is possible to reduce the amount of data needed to represent a sound by sharing this redundant information. One can also

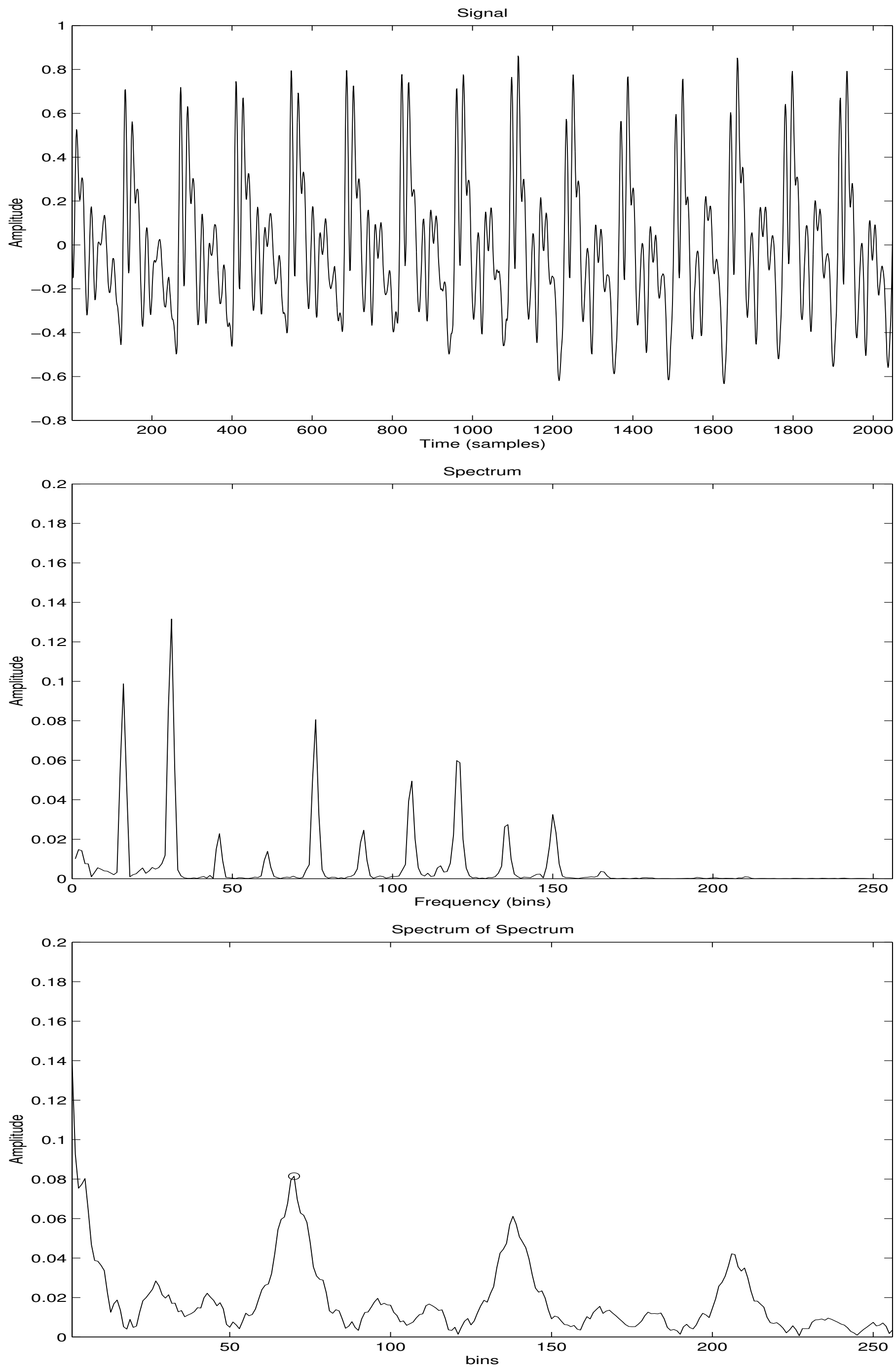


Figure 3.32: Fourier of Fourier. From top to bottom are the original signal (singing voice, sampled at $F_s = 44100$ Hz), its magnitude spectrum, and the magnitude spectrum resulting from the Fourier transform of the previous magnitude spectrum ($N = 2048$, but only the first 256 bins are displayed). One can clearly see in this spectrum the prominent peak corresponding to the fundamental frequency of the original sound.

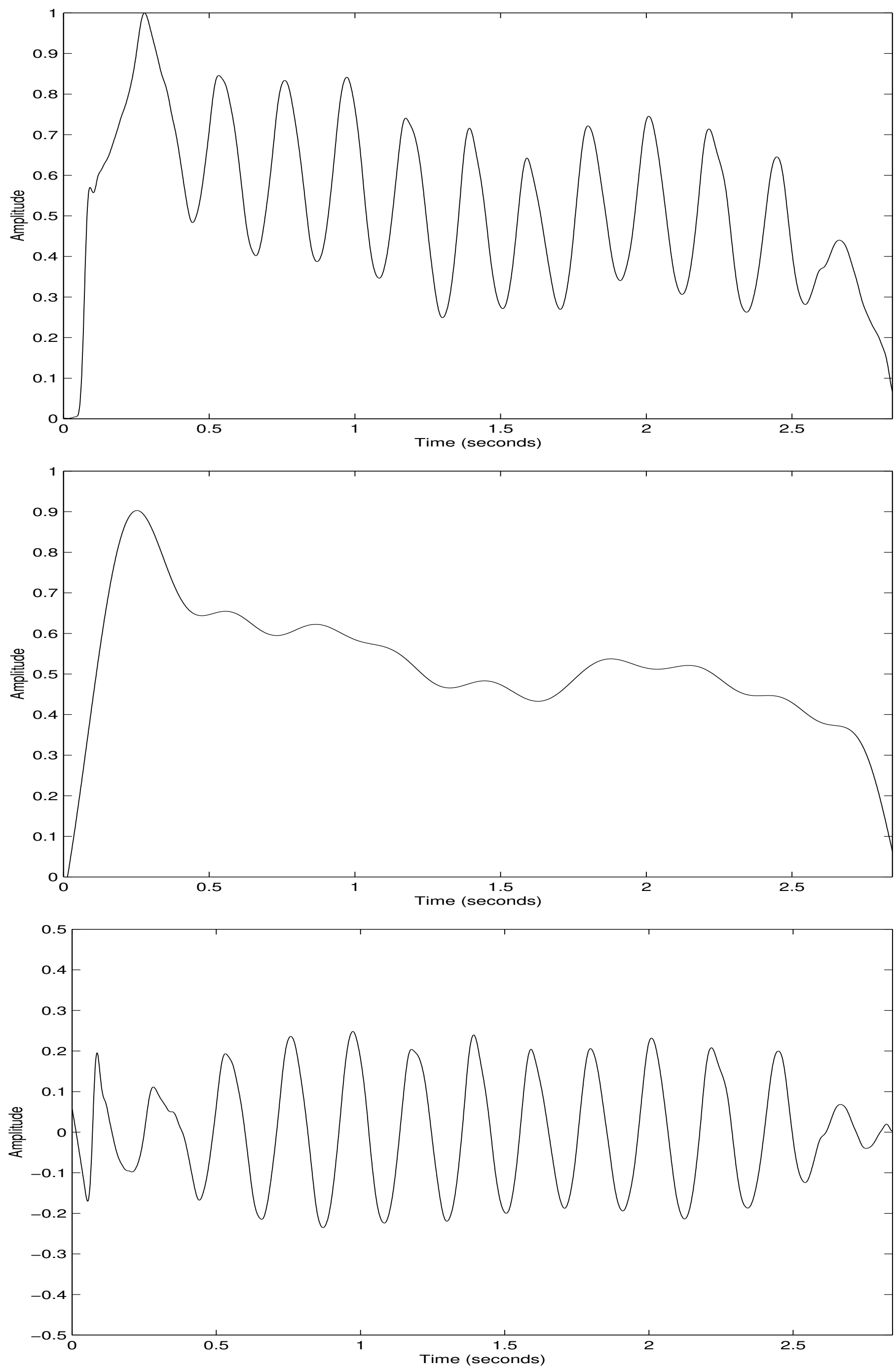


Figure 3.33: The amplitude of the first harmonic of an alto saxophone as a function of time (a_1), decomposed here as a macroscopic envelope ($a_{a_1,0}$) and microscopic variations. From top to bottom are the amplitude function a_1, the associated macroscopic envelope $a_{a_1,0}$, and the residual microscopic variations. As a consequence the tremolo is separated from the envelope.

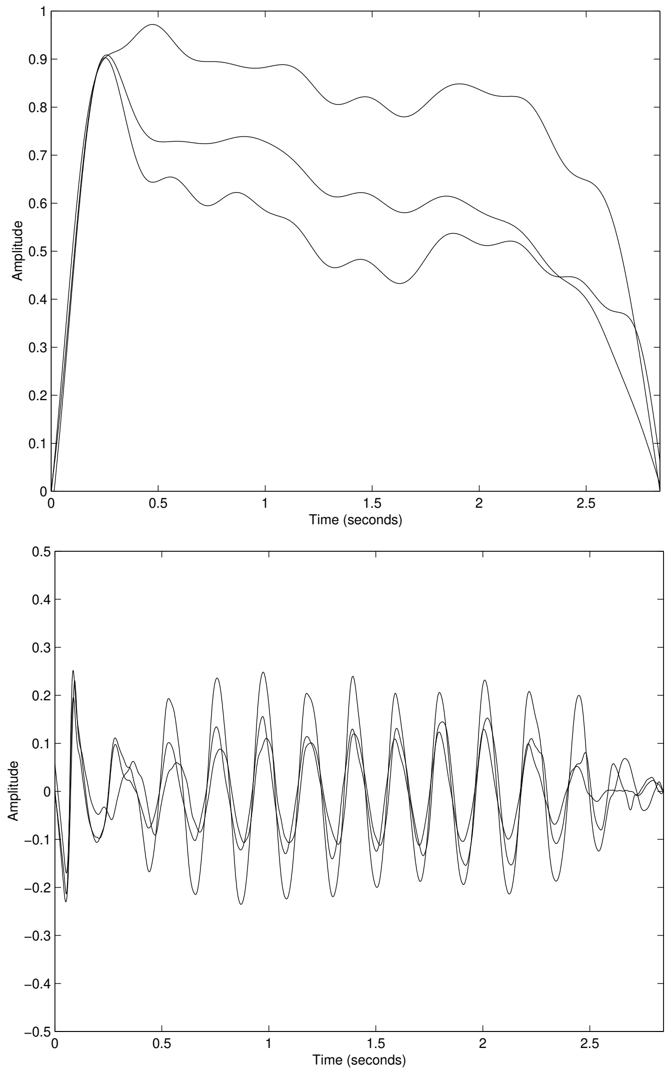


Figure 3.34: Same decomposition as in Figure 3.33, but this time for the first 3 partials.

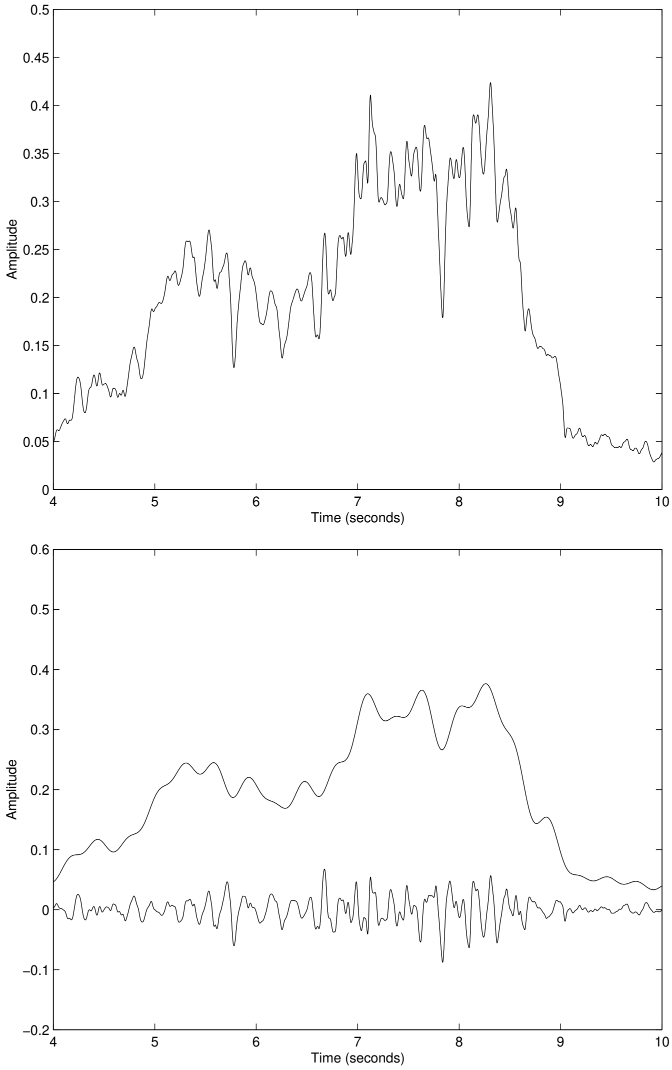


Figure 3.35: The same decomposition as in Figure 3.33, but this time on a singing voice (mezzo female voice). The amplitude of the first harmonic (top) is decomposed into a macroscopic envelope and microscopic variations (bottom).

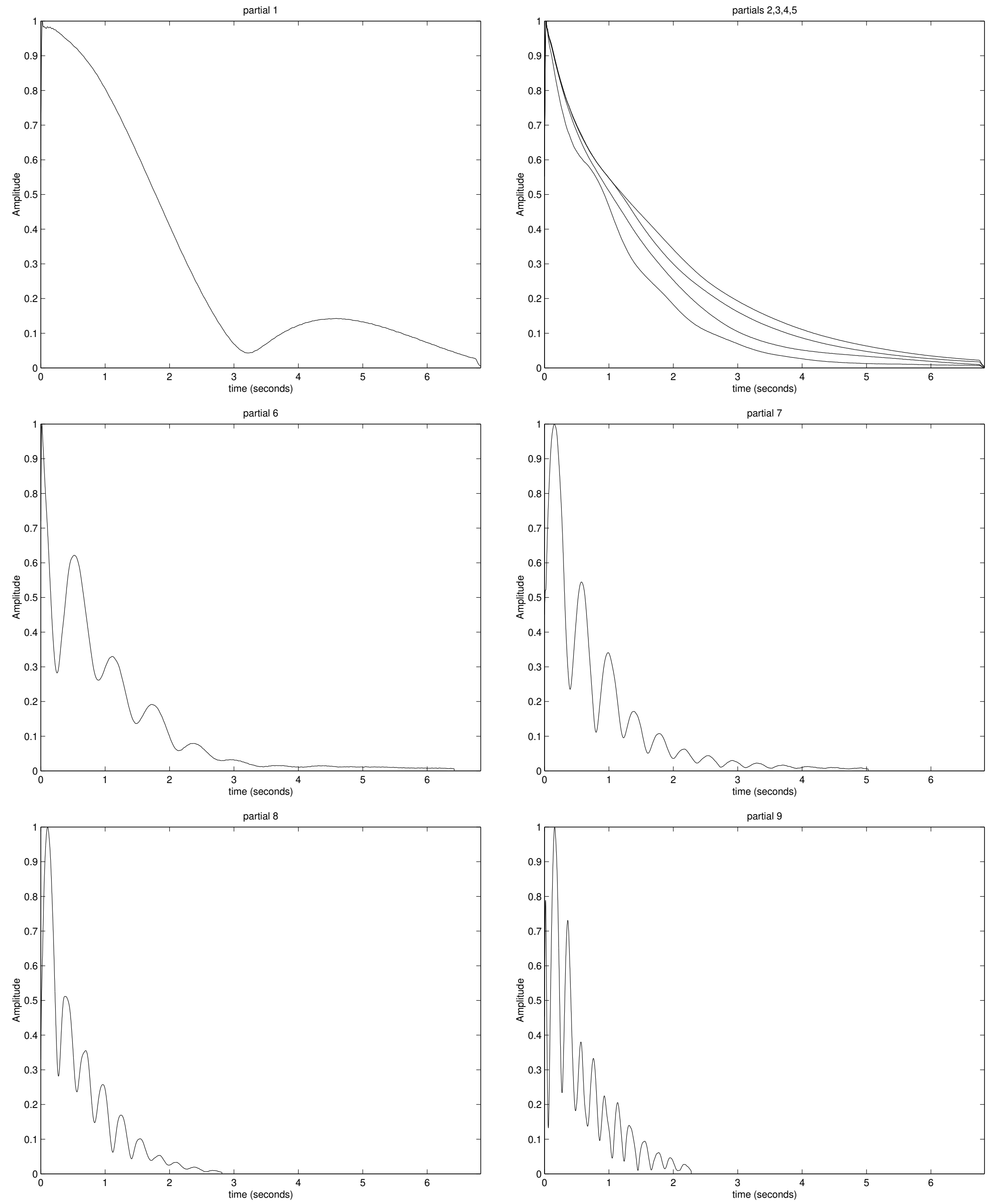


Figure 3.36: The normalized amplitudes of the first 9 partials of a guitar sound. Partials 2, 3, 4, and 5 show very similar evolutions. Higher partials are well-suited for the decomposition illustrated by Figure 3.33.

remove the vibrato or tremolo in a voice, in order to get a “flat” sound, or do the opposite. This possibility of removing or adding the vibrato or tremolo is very useful for clean musical transformations, specially for time-stretching.

3.8.3 Analysis of the SAS Parameters

Regarding the parameters of the SAS model, we can perform the same decomposition as in Equations 3.49 and 3.50 on the A and F parameters. The tremolo and vibrato can thus be extracted from the amplitude A and frequency F, respectively. However, the decomposition of the two-dimensional parameters C and W is much more complicated. We are currently carrying out researches on this topic.

3.8.4 Pitch Tracking

Determining the evolutions with time of the pitch of a sound is an important problem. This is indeed extremely useful for controlling synthesizers from this pitch information and absolutely necessary for pitch-synchronous algorithms such as PSOLA [Pee98].

Various methods have been proposed for the determination of the pitch as a function of time (pitch tracking). They use either the autocorrelation factor [Rab77], other physical [BP93, Lan90] or geometric [CN96] criteria, least-square fitting [Cho97], pattern recognition [Bro92] or even neural networks [SJ89].

We have seen previously that the Fourier of Fourier transform – the magnitude spectrum of the Fourier transform of the magnitude spectrum of the Fourier transform of the signal – is well-suited for pitch tracking, that is for computing the fundamental frequency of the sound, even if it is missing or “virtual”.

We propose to use this Fourier of Fourier transform to perform pitch tracking. We have seen at the beginning of this section that the fundamental frequency of the sound is given by the greatest local maximum of magnitude (apart from the one corresponding to bin 0) in the spectrum resulting from the Fourier of Fourier transform.

The problem is that for some sounds this maximum of energy is detected at the wrong place from time to time. We propose to apply a peak-tracking strategy similar to partial tracking (see Section 3.4), except that this time we deal with “pseudo-partials”, that is partials detected in the spectrum resulting from the Fourier of Fourier transform. Figure 3.37 illustrates this. When two partials overlap at a certain time t – such as P_1 and P_2 in this figure – the partial with the greatest amplitude is said to be dominating. If this partial is longer and louder than the other, we forget the dominated partial. In Figure 3.37, we remove P_3 because it is always dominated by P_2. Once all dominated partials have been removed, we consider the strongest partial, which is the partial who is dominating for the longer period. In Figure 3.37, P_2 is the strongest partial. The frequency of the strongest partial gives the evolutions in time of the fundamental frequency of the initial sound. We have implemented this method in *InSpect* (see Section 3.6), and it has proven to be very accurate in practice.

3.8.5 Advanced Musical Transformations

Even if time-stretching is performed on a spectral sound, artifacts may occur in the presence of vibrato or tremolo. In fact, if the sound is expanded in time, the vibrato and tremolo are slowed down, thus producing a strange musical feeling. But if the sound is compressed in time, the vibrato and tremolo are accelerated, and then artifacts occurs. More precisely, if the

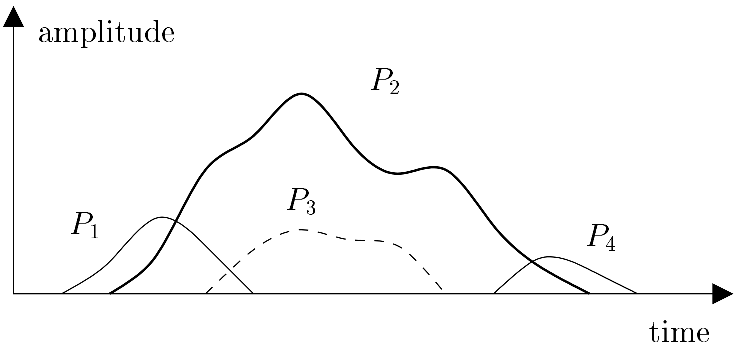


Figure 3.37: The strongest partial (P_2) among the dominant partials (P_1, P_2, and P_4).

original sound has a vibrato at f Hz, if it is time-compressed by a factor k then the vibrato of the resulting sound has a frequency of kf Hz. The vibrato can then exceed the 20-Hz threshold of hearing and thus become audible. This results in a modulation phenomenon.

After Delprat and Arfib [AD99, AD98], we can remove vibrato and tremolo prior to the time-stretching operation and add then the original vibrato and tremolo back into the stretched "flat" sound.

We propose to decompose the evolutions of the partials as in Equation 3.49 and 3.50. Whereas the macroscopic envelope is then time-stretched using the classic method, the microscopic variations are stretched without changing their frequency contents. As a consequence the frequency of the vibrato (or tremolo) is conserved.

3.8.6 About Source Separation

We are also interested in separating sources in the SAS model, that is pseudo-harmonic sources. When several sources are recorded together, the analysis stage produces a set containing their respective partials. Source separation consists in recovering the evolutions of the partials of the different sources, that is, in the SAS model:

$$\{(f_1, a_1), \cdots, (f_P, a_P)\} \to \bigcup_{s=1}^{S} \{(A_s, F_s, C_s, W_s)\} \tag{3.51}$$

More precisely, let us consider the superposition of two SAS sources, so that we have $S = 2$ in Equation 3.51. Since the two sources are nearly harmonic, if all the frequencies of these partials are always very different then it is quite easy to split the set of partials into two subsets – each containing the partials of one of the sources – by considering the arithmetic relations among the partials within the two harmonic sources. But this special case is rather scarce in practice. In fact, there are at least two partials with close – or same – frequencies. Figure 3.38 shows a superposition of two harmonic sources where two partials have exactly the same frequency, and this even happens twice here. In fact, this phenomenon appears almost always within musical chords. Two partials belonging to two different sources are so close in frequency that they lie in the same Fourier transform bin during the analysis stage, as shown in Figure 3.39. This phenomenon is called contamination. The two partials, say p_i and p_j, produce a $V_i + V_j$ complex value in the spectrum at the contaminated bin m. V_i and V_j are the complex values which should have been measured at this bin if, respectively,

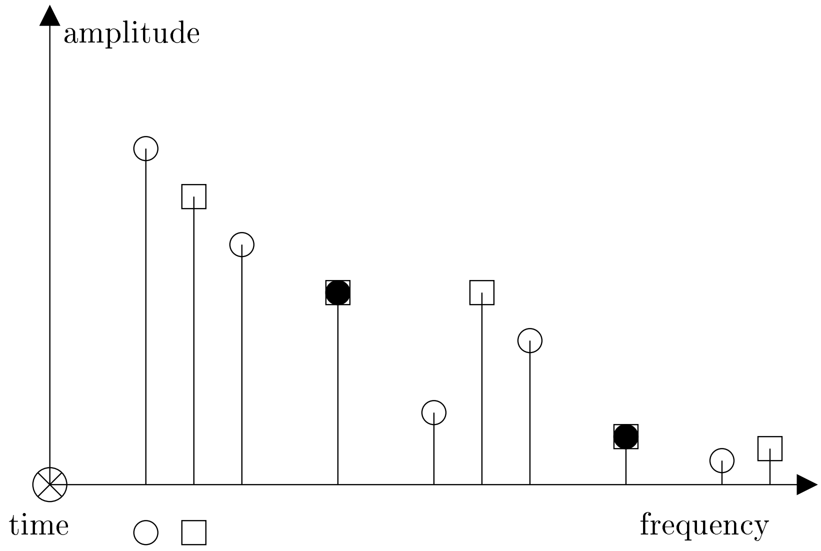


Figure 3.38: The superposition of two harmonic sources. The partials of the first and second sources are indicated with circles and squares, respectively. When two partials have the same frequency, contamination occurs. This phenomenon occurs even twice here.

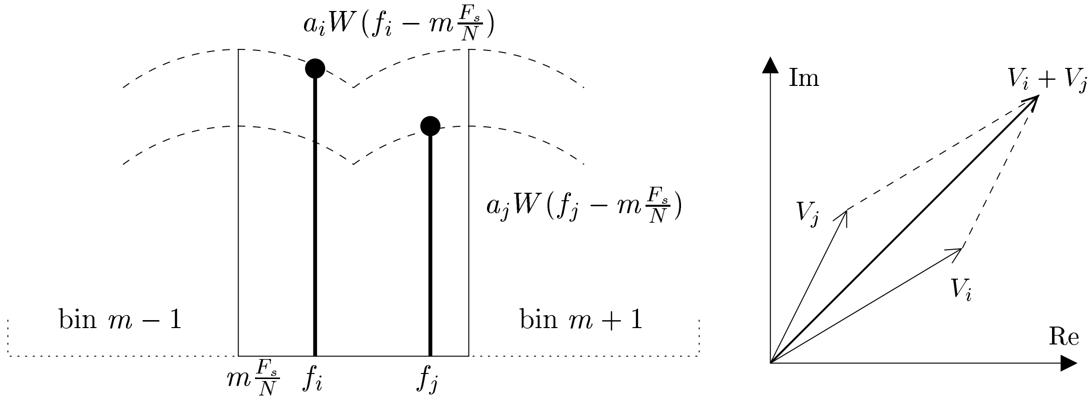


Figure 3.39: The two partials p_i and p_j lie in the same Fourier transform bin m (left). As a consequence they produce a $V_i + V_j$ complex value in the spectrum at the contaminated bin (right). V_i and V_j are the complex values which should have been measured at this bin if, respectively, p_j or p_i had been absent.

p_j or p_i had been absent. More precisely, if W, N, and F_s are, respectively, the (continuous) spectrum of the analysis window, the width of the Fourier transform, and the sampling rate of the analyzed sound, we have:

$$|V_i| = a_i \, W \left(f_i - m \frac{F_s}{N} \right) \tag{3.52}$$

$$|V_j| = a_j \, W \left(f_j - m \frac{F_s}{N} \right) \tag{3.53}$$

The evolutions of the two partials have been averaged into a single partial. The problem is then to recover the evolutions of the two initial partials from this analyzed partial. The idea is to reanalyze the evolutions in time of this contaminated partial. Although we have not investigated it in details yet, we believe that the comparison with the analysis of the evolutions of the uncontaminated partials of each source might help to recover which part belongs to which source.

Chapter 4

Sound Synthesis

The spectral models described in Chapter 2 would be useless in practical applications without an efficient synthesis method able to generate the audio signal from the model parameters, possibly in real time. By synthesis we mean computation of the discrete audio signal in the time domain. Once this discrete signal has been generated, it can be reconstructed in the real – physical – world using a digital-to-analog converter (DAC), as seen in Chapter 2.

As in Chapter 3, the additive synthesis model plays a central role. Additive synthesis requires the computation of a large number of sinusoidal oscillators. Our *InSpect* software tool is able to perform additive synthesis, although this synthesis is not done in real time. Many synthesis methods have been implemented in *InSpect*, thus providing a very convenient way of comparing their respective performances. Section 4.1 presents the most interesting methods for additive synthesis. The fastest method has been implemented in our *ReSpect* software tool. Section 4.2 explains the synthesis algorithm we use, while Section 4.3 describes its implementation in *ReSpect*. *ReSpect* was designed specially for the purposes of real-time spectral synthesis. It is controlled by a flow of additive parameters with a slow rate. Section 4.4 shows how *ReSpect* manages to efficiently up-sample the variations of these parameters using interpolating splines. Section 4.5 explains how psychoacoustic considerations can help reducing the number of partials, thus speeding up the synthesis process. Section 4.6 then presents the synthesis of sounds in the structured additive synthesis model, based on additive synthesis. Finally, Section 4.7 explains the way of synthesizing noise as well.

4.1 Additive Synthesis

In the *InSpect* system (see Chapter 3), we are able to extract very precise information about the partials of a sound, for use with additive synthesis [Moo77], using a high-precision Fourier analysis. The output of such an analysis is a flow of parameter values for a bank of oscillators. Each oscillator in the bank is responsible for exactly one partial. As opposed to methods based on the Fast Fourier Transform (FFT), the frequencies of the oscillators are not fixed, but vary slowly according to the parameter flow. This representation for sound introduced by McAulay and Quatieri [MQ86] is very expressive musically. It has already been successfully used in software packages like Lemur [FH96] and SMS [Ser97b]. Figure 4.1 illustrates this spectral representation. Additive synthesis deals with sounds represented as sums of oscillations. Each oscillation is produced by a sinusoidal oscillator whose frequency f and amplitude a vary slowly over time (see Figure 4.2). Such an oscillator is commonly called a *partial* (see Figure 4.1). We

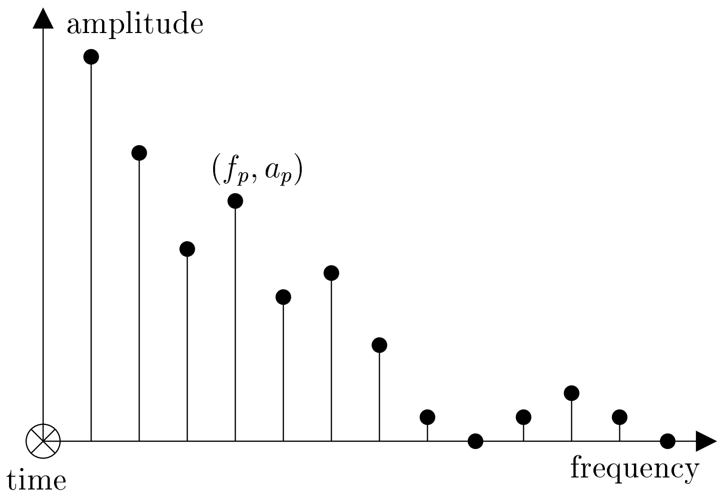


Figure 4.1: Spectral representation, at time t, of a pseudo-periodic sound consisting of 13 partials.

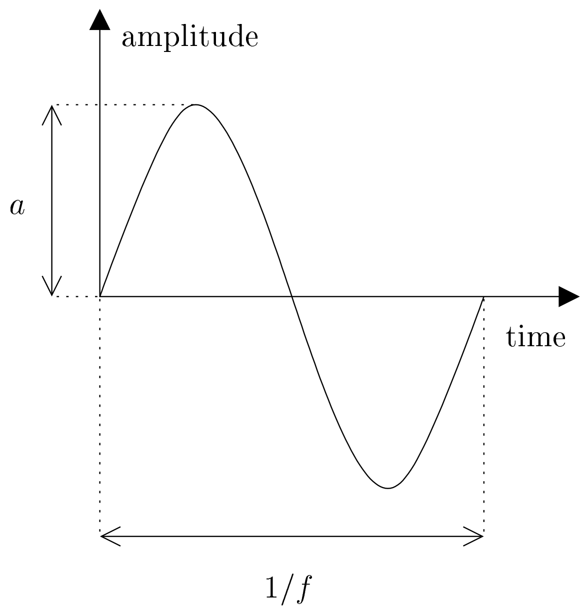


Figure 4.2: One period of an oscillation of frequency f and amplitude a.

need an efficient synthesis algorithm, since the synthesis requires the computation of a large number of oscillators. Always more sophisticated algorithms and the always increasing power of modern computers now allow us to produce hundreds – if not thousands – of simultaneous oscillators in real time, as demonstrated by our *ReSpect* synthesis software tool (see Section 4.3).

We describe systems for generating sounds in real time. The input to the systems is a flow of parameter values. These values control the frequency and amplitude of a bank of oscillators. The time between two sets of parameter values in the flow is much larger than the time between two samples of the resulting sound. The sound generation system must compute the instantaneous value of each oscillator and then sum the results. The remainder of this section will focus on the synthesis of a single oscillator. More formally, the following series of discrete samples must be generated in sequence:

$$s[n] = a\cos(n\Delta_\phi + \phi_0) \quad \text{where} \quad \Delta_\phi = \frac{2\pi f}{F_s} \tag{4.1}$$

The main problem is then to get sufficient performance out of the computation of each instantaneous oscillator value $s[n]$.

4.1.1 Software Oscillators

Of course computing directly the cosine function for each sample is not realistic. This would result in a very accurate but extremely slow synthesis. In fact even the built-in *cos* function of the arithmetic coprocessors of modern general-purpose processors is too slow to suit our needs. This function could be used from time to time, but not for each sample.

The problem, then, is to find a very fast method for generating the sequence of samples for each oscillator with as few operations as possible. There are essentially two possibilities. The first one is based on table lookup and the second is based on incremental computation of a sample based on the previous few samples.

The principle of table lookup is to precompute some values of the sine function, storing them into a table, and to access this table using an index (representing time) at the synthesis stage. However table lookup requires at least 1 addition and 1 multiplication, to update the index for the next sample and to scale the oscillation to the correct amplitude, respectively. In order to achieve a reasonable quality, sine tables either require interpolation among stored values (which is slow) or massive amounts of memory (essentially one table for each frequency).

In the past, table lookup was a reasonably fast method. Memory was relatively fast and arithmetic, especially on floating-point values, was much slower. As processors rapidly became faster and main memory remained roughly the same speed, this technique became less interesting. At the same time, progress was made with respect to the speed of floating-point arithmetic. On a modern processor, a floating-point addition can be done in 1 cycle (pipelined) with a latency of around 3 cycles, and a floating-point multiplication can be done in 1 or 2 cycles with a latency of around 5 cycles (for the Pentium family processors). The trend is towards even faster arithmetic and higher clock speeds whereas memory speed still remains roughly the same.

A natural choice, then, seems to be a method based on incremental floating-point arithmetic, using a recursive algorithm.

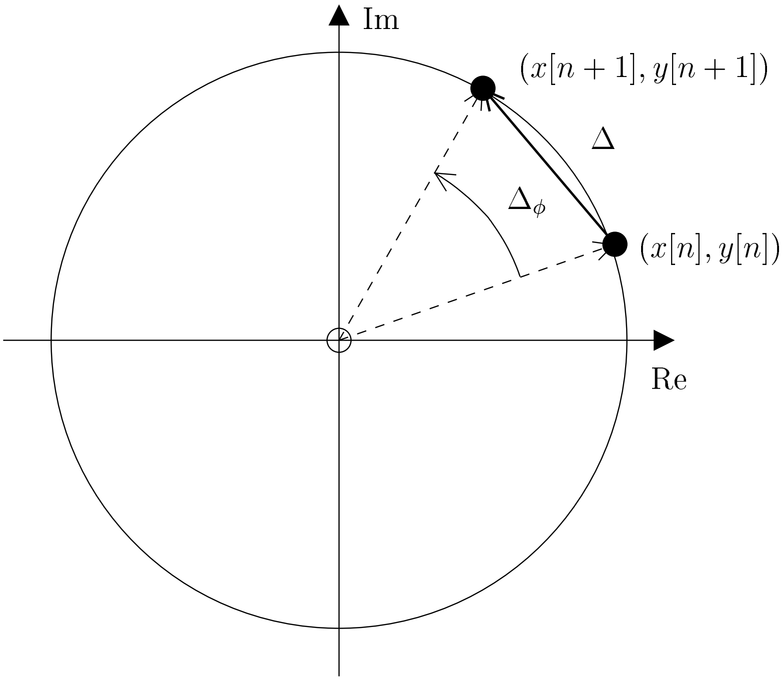


Figure 4.3: The trigonometric circle. V_{n+1} results from the rotation of vector V_n by Δ_ϕ.

Coupled Form

Let us consider complex numbers in the complex plane, and denote by $(x[n], y[n]) = x[n] + j\ y[n]$. As a consequence, the Euler notation is $(\cos(\phi), \sin(\phi)) = e^{j\phi}$ and we have: $s[n] = \mathrm{Re}(V_n)$ with $V_n = a\ e^{j(n\Delta_\phi + \phi_0)}$, where $\mathrm{Re}(x)$ is the real part of the complex number x. Figure 4.3 shows the trigonometric circle in the complex plane. The rotation by Δ_ϕ of the V_n vector is equivalent to a complex multiplication by $e^{j\Delta_\phi}$ of the complex number V_n. More formally:

$$V_{n+1} = a\ e^{j((n+1)\Delta_\phi + \phi_0)} = a\ e^{j(n\Delta_\phi + \phi_0)}\ e^{j\Delta_\phi} = V_n \cdot e^{j\Delta_\phi}$$

This equation is the key of the "coupled form" algorithm (see [GS85, SC92]), which can be defined by this system:

$$\left\{ \begin{array}{lcl} (x[0], y[0]) & = & (\cos(\phi_0), \sin(\phi_0)) \\ (C_x, C_y) & = & (\cos(\Delta_\phi), \sin(\Delta_\phi)) \\ (x[n+1], y[n+1]) & = & (x[n], y[n]) \cdot (C_x, C_y) \end{array} \right. \tag{4.2}$$

i.e.

$$\left\{ \begin{array}{lcl} x[n+1] & = & x[n] \cdot C_x - y[n] \cdot C_y \\ y[n+1] & = & x[n] \cdot C_y + y[n] \cdot C_x \end{array} \right.$$

The iterative computation of each sample $s[n] = x[n]$ of the oscillator requires 4 multiplications and 2 additions, although this computation can be numerically unstable if performed in fixed-point arithmetic.

Magic Circle

The "magic circle" algorithm (see [GS85, SC92]) is numerically stable and requires only 2 multiplications and 2 additions per sample. For $\phi_0 = \pi/2$, it is easy to verify by induction, since $\sin(2u) = 2\sin(u)\cos(u)$, that the samples of s are given by x in the following system (for $\phi_0 = 0$ and using the sine function instead of the cosine one in Equation 4.1 though):

$$\left\{\begin{array}{lcl} x[0] & = & 0 \\ y[0] & = & \cos(\Delta_\phi/2) \\ C & = & 2\sin(\Delta_\phi/2) \\ x[n+1] & = & x[n] + C \cdot y[n] \\ y[n+1] & = & y[n] - C \cdot x[n+1] \end{array}\right. \tag{4.3}$$

Many formulae have been proposed to lower the number of required multiplications and additions. They all take advantage of well-known trigonometric formulae to reduce the complexity. However the resulting methods use only the current value to compute the next one. A good idea is to take advantage of a deeper recursion scheme.

Digital Resonator

Smith uses in [GS85, SC92] the "digital resonator" to perform the incremental computation of the sine function using two previous values to compute the new one with only one multiplication and one addition. This algorithm is optimal in terms of complexity. It is indeed impossible to achieve better results, since an addition without multiplication or a multiplication without addition will produce, respectively, only arithmetic or geometric progressions, far from to the expected sinusoids.

$$s[n] = a\cos(n\Delta_\phi + \phi_0) \quad \text{where} \quad \Delta_\phi = \frac{2\pi f}{F_s}$$

As a consequence, we have:

$$\begin{array}{rcl} s[n+1] & = & a\cos((n+1)\Delta_\phi + \phi_0) \\ \cdots & = & a\cos(n\Delta_\phi + \phi_0 + \Delta_\phi) \\ \cdots & = & a\cos(n\Delta_\phi + \phi_0)\cos(\Delta_\phi) - a\sin(n\Delta_\phi + \phi_0)\sin(\Delta_\phi) \end{array}$$

and

$$\begin{array}{rcl} s[n-1] & = & a\cos((n-1)\Delta_\phi + \phi_0) \\ \cdots & = & a\cos(n\Delta_\phi + \phi_0 - \Delta_\phi) \\ \cdots & = & a\cos(n\Delta_\phi + \phi_0)\cos(\Delta_\phi) + a\sin(n\Delta_\phi + \phi_0)\sin(\Delta_\phi) \end{array}$$

thus

$$\begin{array}{rcl} s[n+1] + s[n-1] & = & 2a\cos(n\Delta_\phi + \phi_0)\cos(\Delta_\phi) \\ \cdots & = & 2\cos(\Delta_\phi)s[n] \\ \cdots & = & C\ s[n] \\ & \text{where} & C = 2\cos(\Delta_\phi) \end{array}$$

Finally, the digital resonator algorithm can be summarized in the following system:

$$\begin{cases} s[0] & = & a\cos(\phi_0) \\ s[1] & = & a\cos(\Delta_\phi + \phi_0) \\ C & = & 2\cos(\Delta_\phi) \\ s[n+1] & = & C \cdot s[n] - s[n-1] \end{cases} \tag{4.4}$$

The incremental computation of each oscillator sample requires only 1 multiplication and 1 addition, but can turn out to be numerically unstable.

4.1.2 Using the Inverse Fourier Transform

In order to efficiently synthesize many sinusoids simultaneously, Freed, Rodet, and Depalle propose in [FRD92, FRD93] to use the inverse Fourier transform, provided that the oscillator parameters vary extremely slowly. The idea is to reconstruct the short-term spectrum of the sound at time t, and then to apply the Inverse Fast Fourier Transform (IFFT or FFT^{-1}) in order to obtain the temporal representation of the sound, and finally to repeat the same computation further in time, thus performing a kind of "inverse phase vocoder".

Spectrum Approximation

The first step of the algorithm is to reconstruct the (short-term) spectrum of the sound at time t from the spectral information given by the partials. More precisely, for each partial its effect in the spectrum has to be reproduced.

Each partial leads to a spectral peak, and more precisely to a maximum of magnitude located in the Fourier transform bin corresponding to its instantaneous frequency. A very important point is that this is the case only if the frequency of the partial is a multiple of the lowest Fourier transform frequency. Most of the time the neighbors of the main bin have significant magnitudes too (see Figure 4.4). The number of neighbors depends on the "analysis" window (w_r), as explained in Chapter 3. Figure 4.5 shows the power spectra of the Bartlett (triangular) and Hann windows. These two windows are good choices for the overlap-add technique described later. In order to reconstruct the sinusoid with an error below -40 dB, 12 bins (main peak and neighbors included) are required for the triangular window, while only 6 bins are necessary for the Hann window. With 12 bins, the error using the Hann window is below -60 dB, which is quite acceptable. Note that apart from the magnitudes, the computation of the phases is also necessary. This can be done in a trivial way by using the zero-phase windowing technique described in Chapter 3, provided that the phase information has been recovered from the partials. Only the first half of the complex spectrum has to be reconstructed, since the other half is its conjugate symmetric, because we expect a real-valued signal after the inverse Fourier transform.

Inverse FFT

When the (short-term) spectrum has been reconstructed, then the Inverse Fast Fourier Transform (IFFT) can be applied in order to obtain the signal back into the time domain.

Many algorithms have been proposed in order to perform the FFT and its inverse as efficiently as possible. Cooley and Tukey propose in [CT65] the well-known radix-2 (or "butterfly") algorithm listed below.

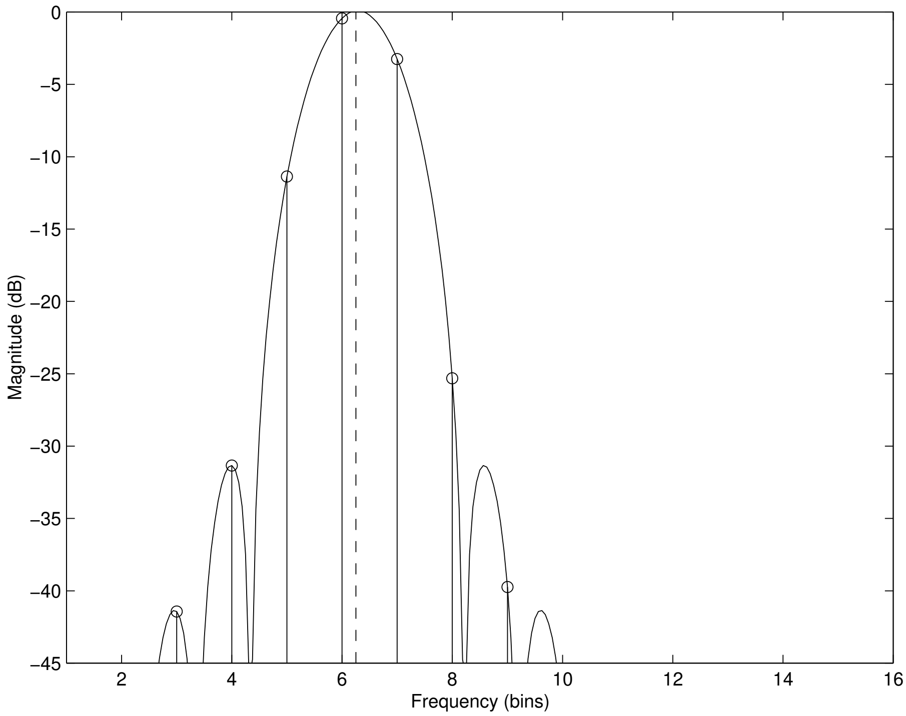


Figure 4.4: When the frequency of the partial is not a multiple of the lowest Fourier transform frequency, the neighbors of the main bin have significant magnitudes too.

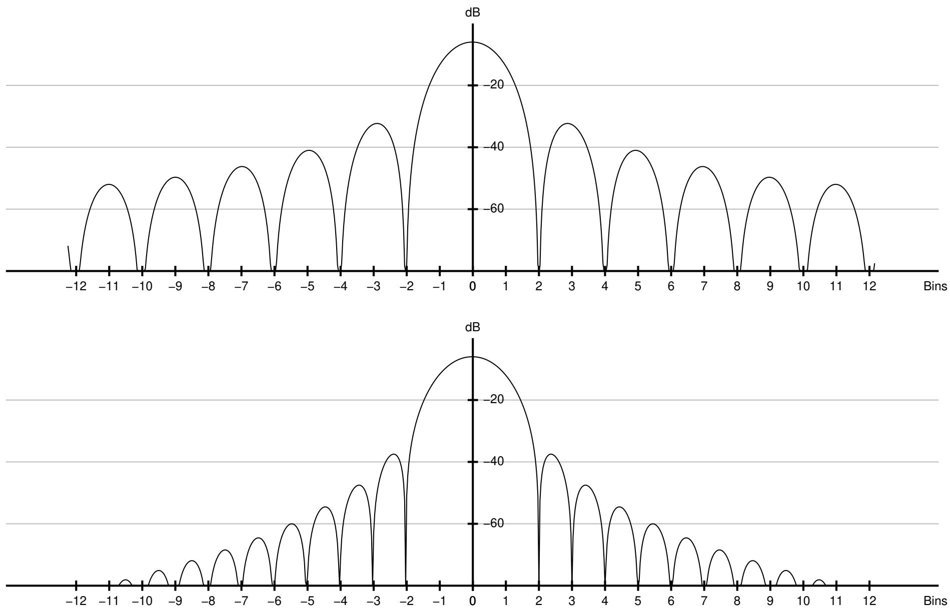


Figure 4.5: Power spectra of the Bartlett (top) and Hann (bottom) windows.

```
void
ifft_transform (COMPLEX *spectrum, REAL *samples, int N)
{
  int i, j, d, k;

  ...

  for (d=1; d<N; d+=d)
    {
      for (i=(N/2), j=0; i>0; )
        {
          butterfly (spectrum, j, d, N);
          i--;
          for (k=1; k<d; k++)
            {
              butterfly (spectrum, j+k, d, N);
              i--;
            }
          j += 2*d;
        }
    }

  ...

}

static void
butterfly (COMPLEX *spectrum, int i, int d, int N)
{
  double x;
  COMPLEX w, tmp;

  x = (double) ((i%d)*(N/(2*d)));
  w.re = cos ( (2*M_PI*x) / N );
  w.im = sin ( (2*M_PI*x) / N );

  tmp.re = (spectrum[i+d].re * w.re) - (spectrum[i+d].im * w.im);
  tmp.im = (spectrum[i+d].re * w.im) + (spectrum[i+d].im * w.re);
  spectrum[i+d].re = spectrum[i].re - tmp.re;
  spectrum[i+d].im = spectrum[i].im - tmp.im;
  spectrum[i].re = spectrum[i].re + tmp.re;
  spectrum[i].im = spectrum[i].im + tmp.im;
}
```

This algorithm is indeed very fast. If N is the size of the Fourier transform (N is a power of 2), the complexity of the algorithm is $O\left(N\log_2(N)\right)$. More precisely, the number of moves, additions, and multiplications is proportional to $N\log_2(N)$. Note also that this algorithm requires $N\log_2(N)$ computations of the *sin* (or *cos*) function, although these computations might be done using the digital resonator, for example. Moreover, at the end of the algorithm the signal is stored as complex numbers so N moves are necessary for extracting the real parts in order to reconstruct the vector of real-valued samples. Another well-known feature of this algorithm is that the complex numbers get stored in bit-reversed order at the end. To recover their real positions, the binary representation of their indices must be mirrored, that is $i = i_0 i_1 \cdots i_{N-1} \rightarrow i_{N-1} \cdots i_1 i_0 = i'$. This results in computation overhead, reasonably small though.

Overlap-Add Technique

The inverse Fourier transform reconstructs small temporal frames of N samples. In order to avoid clicks at the boundaries between successive frames, the overlap-add (OLA) technique [Pee98, PR99] is used. It consists in decomposing the signal into overlapping frames, for example by $N/2$ samples.

Each temporal frame of N samples is multiplied by a weighting window. This window must satisfy the condition that the sum of all the weights of the different overlapped windows must be equal to 1. As a consequence, the most commonly used window is the Bartlett (triangular) window, since it obviously verifies this condition (see Figure 4.8). Figure 4.6 illustrates the overlap-add technique. The temporal frame of signal s is obtained from its spectrum S using the Inverse FFT (IFFT). Then, s is multiplied by the weighting function w. If the reconstruction window w_r is different from the overlap-add window w_{OLA}, then just take $w = w_r / w_{\mathrm{OLA}}$. Finally, the weighted frame is added to the result, and the same algorithm is repeated $N/2$ samples later, and so on. To avoid this multiplication by w for each frame, a good idea is to use the same window for the reconstruction of the peaks in the short-term spectrum prior to the inverse Fourier transform, since the signal frame resulting from this transform will then be already weighted by the appropriate overlap-add window.

We have seen previously that the Bartlett (triangular) window is not very interesting, since it requires the computation of many Fourier transform bins for each partial. Because of the well-known formula of trigonometry $\sin^2(u) + \cos^2(u) = 1$, another (better) candidate is the $\sin^2$ window:

$$w_{\sin^2,N}(n) = \left(\sin\left(\frac{2\pi n}{N}\right)\right)^2 \tag{4.5}$$

However, let us consider the Hann window:

$$w_{\mathrm{Hann},N}(n) = \frac{1}{2}\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right) \tag{4.6}$$

If N is odd, then $N/2 = (N-1)/2$ (since / stands for the Euclidean division among integers). If N is even, just replace $N-1$ by N in Equation 4.6, thus leading to a window very close to the original Hann window. Then we have:

$$\begin{aligned}
w_{\mathrm{Hann},N}(n) + w_{\mathrm{Hann},N}(n+N/2) &= \frac{1}{2}\left[1 - \cos\left(\frac{2\pi n}{N-1}\right)\right] + \frac{1}{2}\left[1 - \cos\left(\frac{2\pi(n+(N-1)/2)}{N-1}\right)\right] \\
\cdots &= \frac{1}{2}\left[1 - \cos\left(\frac{2\pi n}{N-1}\right)\right] + \frac{1}{2}\left[1 - \cos\left(\frac{2\pi n}{N-1} + \pi\right)\right] \\
\cdots &= \frac{1}{2}\left[1 - \cos\left(\frac{2\pi n}{N-1}\right)\right] + \frac{1}{2}\left[1 + \cos\left(\frac{2\pi n}{N-1}\right)\right] \\
\cdots &= 1
\end{aligned}$$

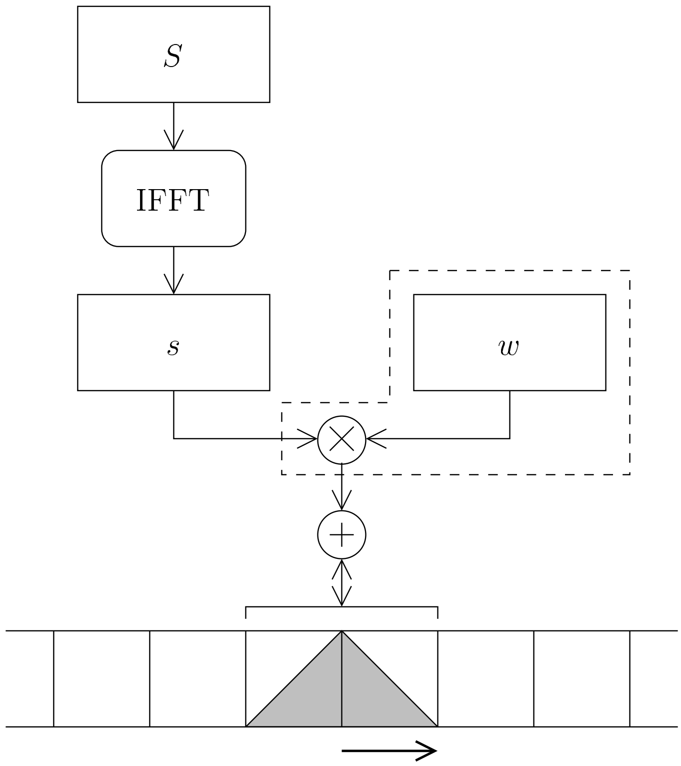


Figure 4.6: The overlap-add technique. The temporal frame of signal s is obtained from its spectrum S using the Inverse FFT (IFFT). Then, s is multiplied by the weighting function w if needed. Finally, the weighted frame is added to the result, and the same algorithm is repeated $N/2$ samples later, and so on.

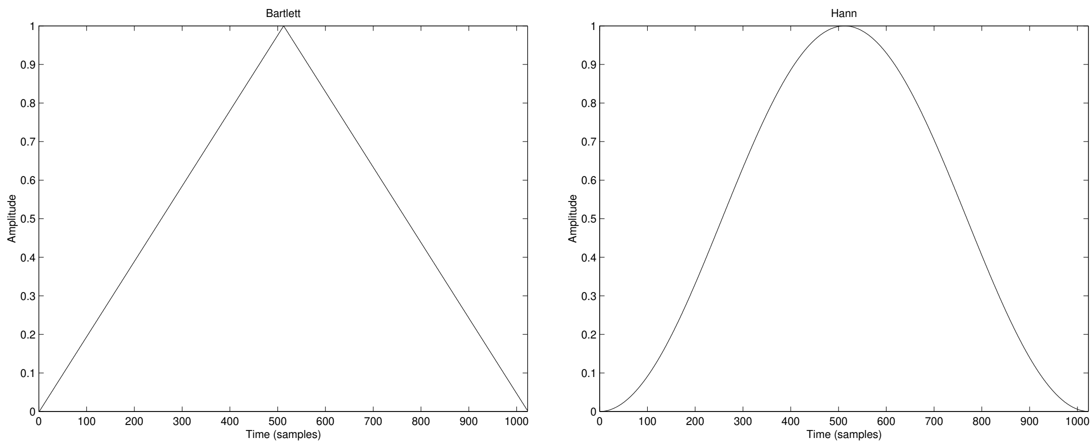


Figure 4.7: Temporal representation of the Bartlett (left) and Hann (right) windows.

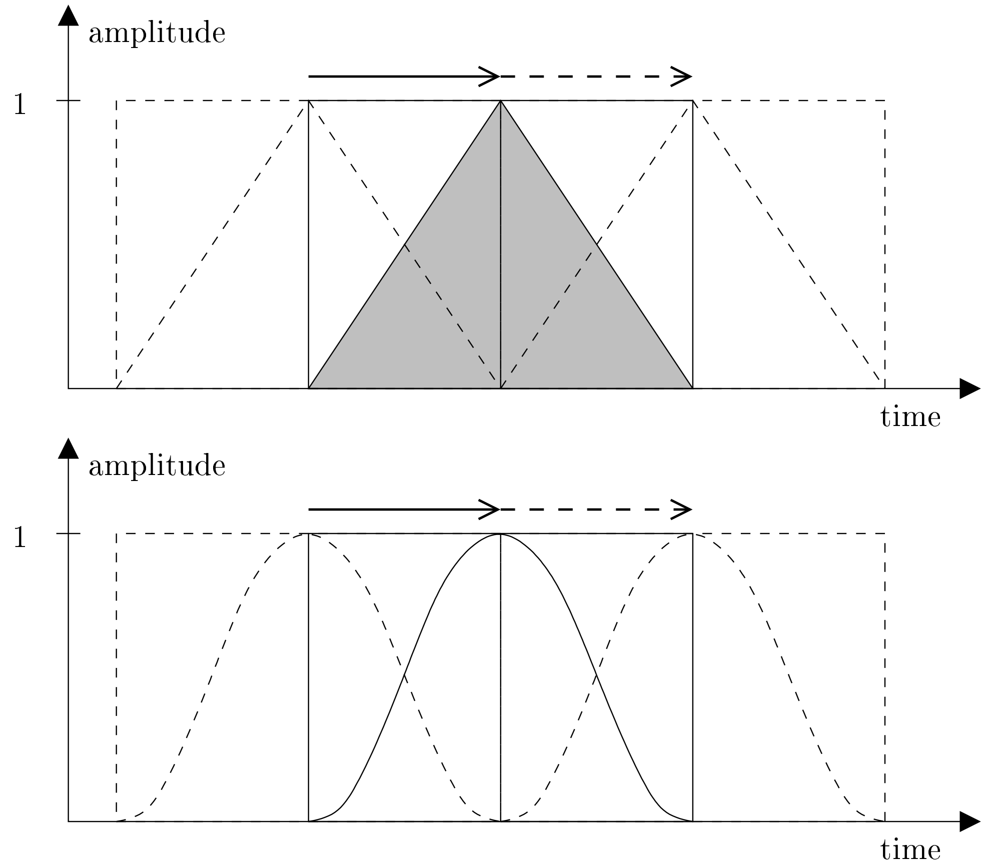


Figure 4.8: Overlap-add with the Bartlett (top) or the Hann (bottom) windows.

As a consequence, this window satisfies the condition that the sum of all the weights of the different overlapped window must be equal to 1 (see Figure 4.8). We propose then to use the Hann window for the overlap-add technique. As seen previously, it is indeed a much better candidate for the reconstruction of spectral peaks.

4.1.3 Comparative Results

We have seen previously that either software oscillators or the FFT^{-1} technique can be used to perform additive synthesis. In order to known which method should be used for a real-time implementation, we propose to compare the respective performances of the digital resonator – the fastest software oscillator – and the FFT^{-1} technique. Two considerations must be taken into account: quality and complexity.

Quality

The digital resonator allows an extremely fine control of each oscillator, since its algorithm can be modified to allow parameter variations (see Section 4.3). On the contrary, the FFT^{-1} technique works only when the oscillator parameters vary extremely slowly, and more precisely when they can be considered as constant within the N samples of the Fourier transform. When these parameters vary over time, then the spectrum gets distorted as explained in Chapter 3, and taking distortion into account will be very costly for the algorithm in terms of computation time. As a consequence, the digital resonator is much more flexible and seems the best choice regarding quality.

Complexity

As a first approximation, we will consider that all the different instructions take approximatively the same computation time.

From the complexity point of view, the digital resonator requires 1 multiplication and 1 addition per oscillator sample, that is $2P$ instructions for P partials.

To achieve sufficient quality with the FFT^{-1} technique, we have seen that the computation of at least 12 spectral bins per partial is necessary. Then, neglecting the moves for complex-to-real conversion and bit-reversal, the inverse FFT requires $kN\log_2(N)$ instructions. Note that the computation is done twice per sample, because of the overlap-add technique. As a consequence, the optimistic number of instructions per sample for the FFT^{-1} method with P partials is $2kN\log_2(N)$. Theoretically determining the exact value of k is extremely difficult because of the features of modern processors such as the caches or pipelines for example. However, in practice, with the Fastest Fourier Transform in the West (FFTW) implementation [FJ98] by the MIT, we measure $k = 30$ on the Intel Pentium II processor. We use FFTW (version 2.1.3) for benchmarking, although the radix-2 algorithm given above is nearly as fast in this case. More precisely, with a processor at 300 MHz, the IFFT takes 2/5 of the time for $N = 256$, and this ratio is proportional to $\log_2(N)$. In the meanwhile, the digital resonator could have synthesized 240 partials, since we show in Section 4.3 that it is able to generate in real time roughly 2 oscillators per MHz of clock speed, that is 600 simultaneous partials on a 300 MHz machine. Note that this time is used by the FFT^{-1} method even if only one partial has to be synthesized, which is not the case with the digital resonator. Of course, when the number of partials increases, the FFT^{-1} method should be more efficient in theory, since for each partial the number of instructions needed to reconstruct its effect in the spectrum is a constant, and not proportional to N as in the digital resonator algorithm.

However things are not so simple in practice, since the reconstruction of a spectral peak (and its neighbors) is very costly. For example, the rectangular or Hann windows require the computation of the *sinc* function, that is the *sin* function computed in a non-incremental way plus a division. Of course other windows could be used, such as the truncated Gaussian one which requires the evaluation of the *power* function…

As a consequence it appears that the digital resonator is more efficient. Freed has probably come to the same conclusion. Indeed, while he was at the origin of the FFT^{-1} method, he proposes together with Hodes in [HF99] to use the digital resonator for additive synthesis as well. At the same time, we were presenting our *ReSpect* software tool for real-time additive synthesis, based on the digital resonator too. The main difference is that we perform all our computations using floating-point arithmetic in double precision, whereas Hodes and Freed use a 16-bit fixed-point arithmetic, and thus have to face numerical imprecision.

4.2 Synthesis Algorithm

Although other methods have been proposed [FRD92, FRD93], we chose to generate each oscillator using the simple recursive description of the digital resonator. It allows us to compute for the discrete signal s the sequence

$$s[n] = a\cos(2\pi f T_s n + \phi_0)$$

for each n incrementally, where a is the amplitude and f the frequency, while $T_s = 1/F_s$ is the sampling period in seconds. Using this formula, the value of some sample $s[n]$ can be expressed as a function of the two previous samples $s[n-1]$ and $s[n-2]$ like this:

$$s[n] = 2\cos(2\pi f/F_s) \cdot s[n-1] - s[n-2] \tag{4.7}$$

As shown previously, this method is indeed more flexible and very efficient. It allows us to have an extremely fine control of each oscillator and the incremental computation of the samples of an oscillator, given any frequency and any amplitude, with only one floating-point multiplication and one floating-point addition per sample, which on most platforms comes to only a few clock cycles.

However, this formula is not very well adapted to our needs. It has two major problems. The first is that the parameters are fixed, whereas they need to vary over time, and the second is that numerical imprecision may cause the values to drift. While the numerical instability could be fixed, the necessity for such parameter adjustments seems to make this formula useless. There are similar formulae that take into account the possibility of linear (and exponential) evolutions of the amplitude, but we are unaware of any fast formula that takes into account linear variation of the frequency as well. Furthermore, if we change from linear interpolation, even those formulae would become useless. In any case, formulae that take into account the variations of the both parameters are much more expensive to compute, at least double the cost of the one above.

The solution we propose in [SM99, MS99] and described in this section is able to adapt the fast formula of the digital resonator so that the parameters may evolve and that drift is avoided, thus leading to a high-quality and very efficient synthesis technique since our solution represents only a very small percentage overhead compared to the original digital resonator.

The algorithm is based on two crucial observations. First, our experiments indicate that abruptly changing the amplitude by a reasonable amount does not introduce any audible noise or distortion. Similarly, changing the frequency by a reasonable amount also does not introduce any audible impact on the quality of the sound. Notice that an abrupt change in phase might have such an impact. But as long as the phase stays the same, we may change the frequency by quite a lot. The second observation is that the slope of our parameter values, i.e. their speed of variation, is going to be relatively small compared to what would still go undetected.

With this in mind, we can now present our method. Based on two consecutive parameter values, determine the slope of the evolution of the parameter values. The slope is defined to be the difference in parameter values divided by the number of signal samples between two parameter points. In our case, there can be up to 1000 samples in such an interval for a sampling rate of 44100 Hz. From experimental data, we know the maximum allowable difference in parameter values we can have without creating any audible distortion or noise. Divide this value with the slope previously obtained. The result of the calculation is the number of samples that can be generated without any adjustment of the parameters.

Experiments show that we frequently get intervals of 100 samples or more between necessary adjustments of parameter values. Thus, even if the computation to adjust parameter values is considerably more expensive than that of generating a sample, we are still within a few percents of extra cost compared to maximum speed when parameters do not evolve at all.

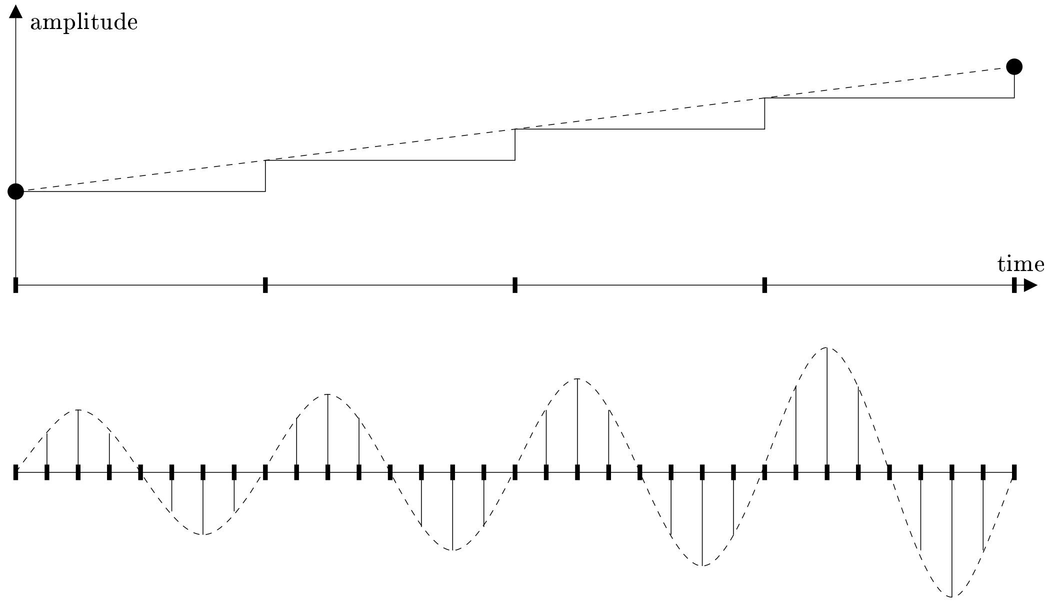


Figure 4.9: Variation of the amplitude of an oscillator and the resulting audio signal. Between two parameter changes, some interpolated values are computed (4 in this example). And between two interpolated values, many samples are computed.

4.2.1 Changing the Control Parameters

The additive synthesis parameters vary over time. In theory, both frequency and amplitude can vary slightly between each new computed sample. The reason is that parameter evolution is expressed as an interpolation between values in the parameter flow. So even though we compute thousands of samples between two parameter values in the flow, because we interpolate, we must be prepared to adjust both the frequency and the amplitude of our oscillators at each sample.

In fact, as mentioned above, we do not need to adjust the parameters at each sample, but only every 100 samples or so. In the current implementation of *ReSpect*, the frequency and amplitude of each oscillator is updated every 64 samples at 44100 Hz. Figure 4.9 shows how our method handles the variations of the parameters. The parameters of the partials are sent to *ReSpect* about 172 times per second, which corresponds to a period of 256 samples at 44100 Hz, and the parameters of the oscillators are adjusted every 64 samples. For now we may assume that the instantaneous values of the parameters are found by linear interpolation between the values present in the flow. Other systems are possible. In particular, we propose in Section 4.4 to consider the parameter values as the samples of a continuous-time signal with a frequency of at most half the one defined by the time between parameter values.

Now let us focus on the way the parameters of the oscillators are changed (every 64 samples). Changing the amplitude is obvious. It is just a matter of changing a, provided that this value is known. The amplitude can be changed from a_1 to a_2 by a simple multiplication of the two last values of the oscillator by a_2/a_1. Changing the frequency is easy too, since it is only a matter of recomputing $2\cos(\Delta_\phi) = 2\cos(2\pi f/F_s)$ with the new value of f. The phase must not be changed since it must remain continuous.

Given 3 previously-computed values of the oscillator:

$$\begin{aligned} s[n-1] &= a\cos(\phi - \Delta_\phi) \\ s[n] &= a\cos(\phi) \\ s[n+1] &= a\cos(\phi + \Delta_\phi) \end{aligned}$$

it is possible to recover the amplitude a and the phase ϕ of the sine at sample n. Indeed, we have:

$$\begin{aligned} s[n-1] - s[n+1] &= 2a\sin(\phi)\sin(\Delta_\phi) \\ ((s[n-1] - s[n+1])/2)^2 + (\sin(\Delta_\phi)s[n])^2 &= a^2(\sin(\Delta_\phi))^2 \end{aligned}$$

and finally:

$$a = \sqrt{\left(\frac{s[n-1] - s[n+1]}{2\sin(\Delta_\phi)}\right)^2 + (s[n])^2} \tag{4.8}$$

Since $\Delta_\phi = 2\pi f/F_s$ with $0 < f < F_s/2$, we have $0 < \Delta_\phi < \pi$ and $\sin(\Delta_\phi) \neq 0$. Then, recovering the phase is easy:

$$\phi = \arccos\left(\frac{s[n]}{a}\right) \tag{4.9}$$

Finally, changing the oscillator parameters can be done using this algorithm:

1. Get the amplitude using Equation 4.8. Of course we may store the current amplitude of each partial to avoid recomputing it, provided that no drift – due to numerical instabilities – has occurred;

2. Either just scale the last 2 values of the oscillator to match the new amplitude while keeping phase continuity, or recover the phase from the last samples using Equation 4.8 and recompute the two initial values according to this phase and the new amplitude.

4.2.2 Avoiding Discontinuities

We already mentioned that it is possible to adjust (for instance) the amplitude by a reasonable amount without introducing any distortion. But we can actually do even better.

Amplitude

Such amplitude adjustments are more audible the greater the absolute value of the current sample to be computed. The reason for this is that the difference in amplitude is multiplied by the value of the signal:

$$a_1\cos(\phi) \to a_2\cos(\phi)$$

The smaller the signal, the smaller the (potentially audible) difference in the amplitude difference. This phenomenon is shown in Figure 4.10. The idea is then to change the amplitude at the right moment to guarantee the continuity of the signal. The best time for changing a from a_1 to a_2 is when $\cos(\phi)$ is close to zero. More precisely, the best and worst cases are, respectively:

- best case: $\phi \equiv \frac{\pi}{2}\ (\pi)$
- worst case: $\phi \equiv 0\ (\pi)$

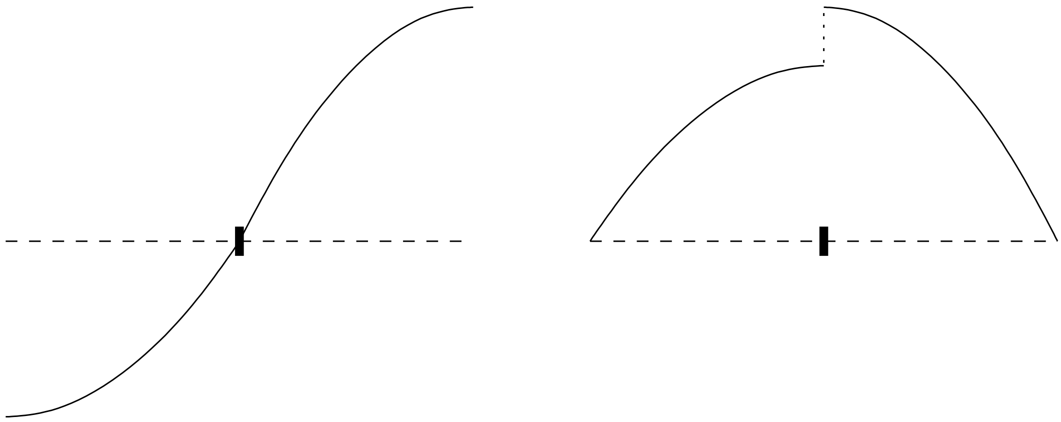


Figure 4.10: Changing the amplitude either when the signal is minimal (left) or maximal (right). It appears that the left case is much better, since it avoids amplitude discontinuities (clicks).

Frequency

The frequency parameter has the inverse problem.

$$\frac{d\phi}{dt} = 2\pi f_1 \to 2\pi f_2$$

An abrupt change in the frequency is more likely to be audible when the value of the signal sample is small. The reason for this is that the derivative of the signal changes as a result of a change in frequency, and the derivative is zero when the absolute value of the signal is the greatest. This phenomenon is shown in Figure 4.11. The idea is then to change the frequency at the right moment to guarantee the continuity of the first derivative of the signal. Since we have

$$\frac{d(a\cos(\phi))}{dt} = -a\frac{d\phi}{dt}\sin(\phi)$$

the best time for changing f from f_1 to f_1 is when $\sin(\phi)$ is close to zero. More precisely, the best and worst cases are, respectively:

- best case: $\phi \equiv 0 \ (\pi)$
- worst case: $\phi \equiv \frac{\pi}{2} \ (\pi)$

Enhanced Algorithm

With this in mind, we can improve the situation even further. The idea is then to delay changes in amplitude until the signal value is close to zero, and to delay changes in frequency until the absolute value is maximal. This algorithm allows us to obtain performance very close to the maximum theoretical value for fixed parameter values while providing a higher synthesis quality.

Naturally, if we had to test the value in each iteration to determine whether it is close to zero or, on the contrary, close to maximal, we would lose at lot of computational power. Such a test would take time comparable to the computation of a sample, which would slow us down by a factor close to 2.

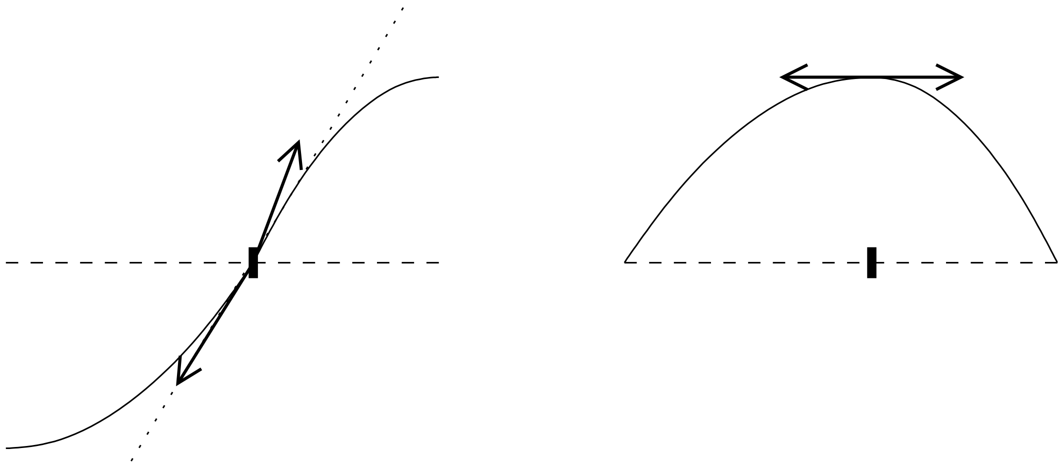


Figure 4.11: Changing the frequency either when the signal is minimal (left) or maximal (right). It appears that the right case is better, since it avoids derivative discontinuities (clicks again).

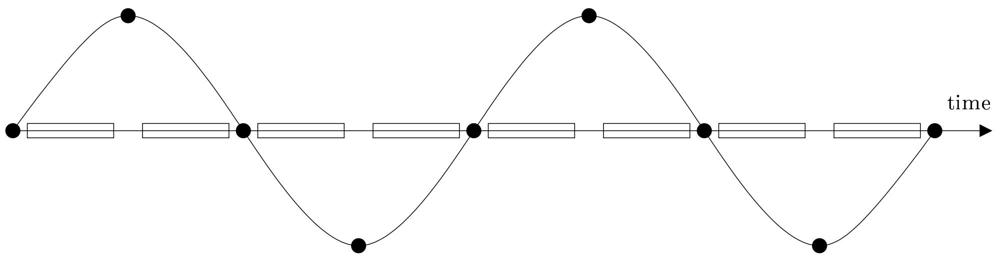


Figure 4.12: Algorithm loop. Either the amplitude or frequency are changed (points), then some oscillator samples are computed (blocks).

Fortunately, we can avoid that problem, and here is how. When parameters are adjusted, we precompute the number of iterations before the next update is necessary. This computation makes sure that the phase after the next block of iterations is optimal. By optimal we mean that it should be close to 0 or π for the frequency and close to $\pi/2$ or $3\pi/2$ for the amplitude. The computation of the samples is then started for a known number of iterations. When that computation finishes, we know it is the optimal time to adjust one of the parameter.

To avoid a test for the loop counter in each iteration, the loop is unfolded a sufficient number of times so that the time taken by this test is negligible.

Finally, the enhanced algorithm consists in forever repeating the following loop, illustrated in Figure 4.12:

- May change either amplitude or frequency;
- **Compute some oscillator samples**;
- May fix oscillator instability.

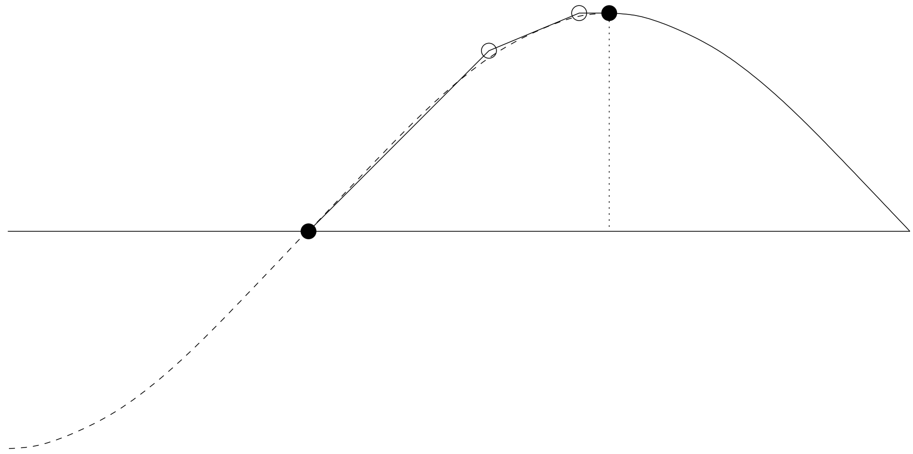


Figure 4.13: Approximation of a sinusoid with a piecewise-linear function consisting of 12 points. This approximation is shown on a quarter of a period, so only 4 points are displayed. The original values of the cosine function as well as its first derivative are conserved for the phases which are integer multiples of $\pi/2$ (black points).

4.2.3 Synthesizing Low Frequencies

The problem with low frequencies is that the optimal update times for the parameters are too scarce. For example, an oscillator with $f = 50$ Hz and a sampling rate of $F_s = 44100$ Hz has update times occurring every 441 samples, which is much more than the minimal update period of 64 samples decided above.

We envisage to address this problem while reducing the complexity of the algorithm at the same time. The idea is to approximate the low-frequency sinusoids by piecewise-linear functions. It is indeed possible to make such an approximation, provided that enough points are used. Figure 4.13 suggests that only 12 points per period are sufficient of a decent quality. The update times correspond then to these points, which are close enough in time to guarantee frequent parameters updates, and distant enough to preserve the performance of the algorithm, which in the case of a linear approximation requires only one addition – and no multiplication – per sample. When the parameters are changing, the next target point in the approximation is computed with the new parameters, and reached by linear interpolation as usual, thus guaranteeing signal continuity.

4.2.4 Numerical Imprecision

It is well known that the fast formula that we use for generating samples does not behave very well with respect to loss of precision in numerical calculations. If iterations are carried out for a long time, drift in the value of the frequency can cause the sound to be severely distorted.

Numerical stability is not a real problem since we can measure and adjust the parameters regularly and often enough to avoid any drift, and rarely enough to preserve performance. Since our method requires us to adjust our parameters every so often, say every 100 samples or so, we may take advantage of this periodicity to recompute our initial samples from scratch, i.e. using trigonometric functions. This way, we would not only adjust our parameters, but also compensate for any possible drift due to numerical imprecision. Should computing the trigonometric functions be expensive at this frequency, it can be done at a much lower frequency than is required to adjust parameters. It just happens to be convenient to do both at the same time.

Fortunately, distortion phenomena will only appear after a considerable number of iterations, in particular since we carry out our calculations using floating-point arithmetic in double precision. We have tested the numerical precision for 10^9 samples, corresponding to a partial length of more than 6 hours at 44100 samples per second. No amplitude drift was measured, even with a very small amplitude of 10^{-6}, corresponding to -120 dB (the least audible amplitude). A drift in phase was measured, but this is not a problem since only the frequency matters to our ears.

In fact the problem is only with extremely small amplitudes. A solution is either to force the partial to die when its amplitude is too low or to compute a "normalized" resonator – that is with its amplitude equal to 1 – and use another multiplication to scale the amplitude to a. Both solutions have been tested. Since the digital resonator using floating-point arithmetic in double precision is stable enough, it is better to use the first solution and avoid this multiplication. In the second case, the synthesis of the 64 oscillator samples between the parameter adjustments is (in C programming language):

```
c = 2 * cos ((2*M_PI*f)/SAMPLING_RATE);

for (i=0; i<STEP; i++)  /* STEP is 64 */
  {
    double snew = sn * c - sn_1;
    *samples++ += a * sn;  /* this multiplication could be avoided */
    sn_1 = sn;
    sn = snew;
  }
```

4.3 *ReSpect* Software Package

ReSpect ("Respect Spectrum") [MS99, Mar00b] is a sound synthesis program specially designed in order to perform the real-time additive synthesis of sounds from their spectral representation. It is mainly an implementation of the synthesis algorithm described in Section 4.2. *ReSpect* can generate many oscillators simultaneously and behaves as a virtual sound card accepting spectral sounds. *ReSpect* is freely distributed [Mar00b] according to the GNU General Public License (GPL) [FSF91].

4.3.1 Implementation

ReSpect is a module for the Linux operating system and works on top of the free version of the Open Sound System (OSS) [wwwb] included in the kernel source tree at the moment. The problem with OSS is that it does not support recent sound cards. We expect to switch to the Advanced Linux Sound Architecture (ALSA) [Kys00, wwwa] very soon. The problem is that ALSA lacks some low-level functionalities which are required by our module. For example, *ReSpect* must be able to open the hardware sound card, and to install a call-back function in order to be warned when the output buffer of the sound card is getting empty, in order to avoid clicks in the output signal.

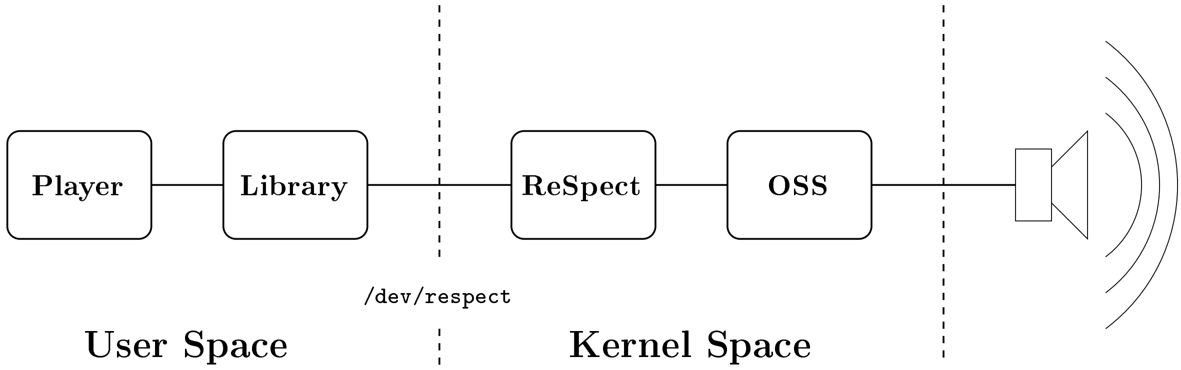


Figure 4.14: Overview of the *ReSpect* architecture.

The aim of the *ReSpect* module is to provide the user with a virtual sound card accepting spectral sounds. Since *ReSpect* runs in the kernel space, it does not depend on the scheduler of the user space. As a consequence it can perform the real-time synthesis of spectral sounds with high priority and low latency. Since *ReSpect* has a direct hardware access, we can guarantee that there will be no click during a live performance using *ReSpect*. In the worst case we will hear a steady sound for a few milliseconds if the spectral information does not arrive on time at the device driver.

4.3.2 Device Driver Protocol

When *ReSpect* is installed as a module in the UNIX kernel, the resynthesis is controlled by writing in a special device driver: `/dev/respect`. This driver maintains an ordered list of the active partials currently synthesized. Each partial p is represented by a (f_p, a_p) pair of floating-point numbers in double precision. Recall that the functions f_p and a_p are the frequency and amplitude of the p-th partial, respectively. The number of synthesized partials can vary over time. When a partial dies, the special $(0, 0)$ pair is placed at its position in the partials list. After that, the partial does not exist anymore. When a new partial appears, its is simply appended to the partials list. When the pairs are written into the device driver, the following protocol must be respected in sequence:

1. For each partial p, write its (f_p, a_p) pair.
   If a partial dies, the $(0, 0)$ pair is written.

2. For each new partial, write its pair.

3. Write the $(-1, -1)$ end-marker pair.

This steps should be repeated for each temporal frame, at a frequency depending on the sampling period. Since we are in the spectral model, the frequency corresponding to this period can be as low as 40 samples per second. Although it will be generalized in the near future, *ReSpect* currently accepts only rates of $44100/(64 * 4) \approx 172$ and $44100/64 \approx 689$ samples per second.

Let us denote by $P(n)$, $B(n)$, and $D(n)$ the number of partials, respectively, present, born, and dead during the n-th frame. We have $P(n+1) = P(n) - D(n) + B(n+1)$. An incoherent number of partials in a frame is a clue to protocol violation and in this case the device closes automatically. Here is an example respecting the device protocol:

```
{
  int    audio;
  int    i, j;
  double end_marker = -1;

  partial *active[MAX_NUMBER_OF_PARTIALS];
  int      nactive = 0;

  /* open the device of ReSpect */

  audio = open ("/dev/respect", O_WRONLY);

  /* play n frames */

  for (i=0; i<=n; i++)
    {
      int      nclosed;
      partial *dst, *src;

      /* add new partials (births) */

      { for each new partial p, do: active[nactive++] = p; }

      /* play active tracks */

      nclosed = 0;

      dst = src = active;

      for (j=0; j<nactive; j++)
        {
          double  f, a;
          partial p = *src;

          if ( death of partial p )
            {
              f = 0;
              a = 0;
              nclosed++;
            }
          else
            {
              f = ( frequency of p at time i )
              a = ( amplitude of p at time i )
```

performance	Pentium II	Pentium III
standard code	**2**	> 4
pipeline optimization	3	> 6

Table 4.1: The number of partials that can be synthesized in real time by *ReSpect*. The values are given in partials per MHz of CPU clock. The sampling rate of the synthesized sound is $F_s = 44100$ Hz.

```
            *dst++ = p;  /* p still active */
          }

        src++;

        write (audio, &f, sizeof(double));
        write (audio, &a, sizeof(double));
      }

    write (audio, &end_marker, sizeof(double));

    nactive -= nclosed;
  }

  /* close the device of ReSpect */

  close (audio);
}
```

4.3.3 Performances

The method for fast additive synthesis we have presented above is indeed very efficient. The following performances are given for the *ReSpect* implementation for Linux PC using the Intel Pentium processor. We have successfully obtained close to optimal performance. Currently, with this implementation, we are able to generate roughly 2 oscillators per MHz of clock speed on Intel Pentium II processors. In other words, we obtain the simultaneous generation of around 800 oscillators in real time on a 400 MHz machine. At the time these lines are written, the Pentium III 800 MHz is available, and the 1 GHz machines are announced. As a consequence, for most sounds in the additive synthesis model, our system is already capable of faithfully resynthesizing signals from their spectral representations in real time. However, we keep on optimizing *ReSpect* and its synthesis algorithm.

By switching to recent processors, we could take advantage of SIMD (Single Instruction, Multiple Data) architectures. For example the Pentium III has news instructions that allow the simultaneous computation of 4 floating-point additions or multiplications. In theory the algorithm should be 4 times faster, although we have not verified this in practice yet. The problem with these new Pentium instructions is that they work in single precision. To adapt them for double precision – needed by the digital resonator – we lose half of the speed. Table 4.1 gives the number of partials that can be synthesized in real time by *ReSpect*. The speed

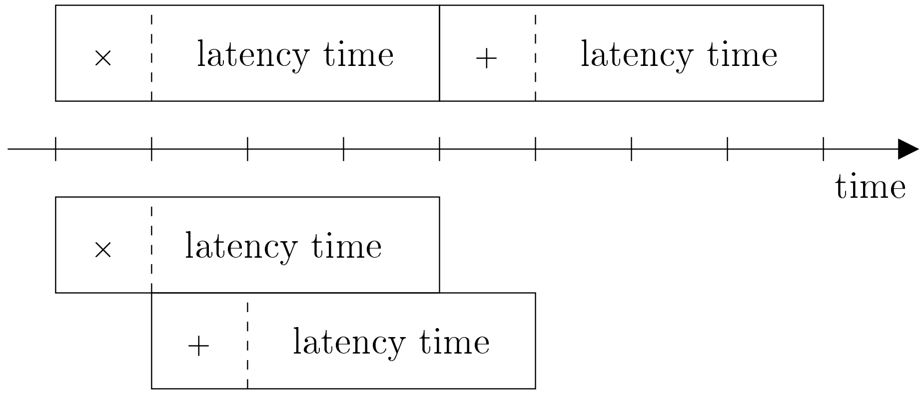


Figure 4.15: Taking advantage of the pipeline can reduce the computation time. Example of non-pipelined instructions (top) and pipelined ones (bottom). Ticks on the time axis correspond to processor cycles.

of the computation can be increased by optimizing the use of the pipeline of these processors. Figure 4.15 illustrates this. However the resulting algorithm is very processor-dependent, and the optimization of the algorithm for a wide variety of processors would be a tough work.

4.4 Fast Resampling Using Interpolating Splines

The additive synthesis parameters are slow-varying functions of time. More precisely, these functions can be regarded as (control) signals which are band-limited in frequency with a maximal frequency under the lowest audible frequency, that is 20 Hz approximatively. As a consequence we do not assume a linear variation of the parameters. We intend to do even better to obtain a high quality with a low rate for the parameter flow. Considering the parameter values as the samples of a band-limited signal allows us to reduce the frequency of the updates of the parameter flow of the device. Since the maximal frequency is under 20 Hz, in theory sampling the parameters 40 times per second or so is sufficient (see Chapter 2). In practice we use a value of about 172 times per second, in order to be able to encode fast variations for partials with high frequencies. Psychoacoustic experiments show that the maximal frequency of the control signal is not a constant, but a function proportional to the frequency of the controlled signal. More precisely, to avoid modulation phenomena, the maximal frequency of the control signal is about 0.35% of the frequency of the controlled partial (see Chapter 1), that is 70 Hz for 22 kHz (considered as the highest audible frequency).

4.4.1 Resampling Process

The control parameters are sent to the device only every 256 samples of the output signal. Since the parameters of the oscillators must be updated every 64 samples, the discrete signals of the control parameters must be up-sampled by a factor 4.

As mentioned in Chapter 2, this up-sampling can be done using a reconstruction filter whose impulse response r is a *sinc* function multiplied by a bell-shaped window w, for example the Hann window. This typical shape for the impulse response was called “Mexican hats” in

Chapter 2. More formally, we have:

$$r(t) = w(t)\mathrm{sinc}(t) \tag{4.10}$$

with

$$\mathrm{sinc}(t) = \frac{\sin(\pi t)}{\pi t} \quad (\text{and } \mathrm{sinc}(0) = 1) \tag{4.11}$$

The discrete Hann window of size N is defined by equation:

$$w_{\mathrm{Hann,N}}[n] = \frac{1}{2}\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right) \tag{4.12}$$

Using a convolution would severely degrade the performance of our synthesis algorithm. The problem is then to compute the intermediate values of the parameters in an efficient way, since we do not intend to use a reconstruction filter as in Chapter 2. We propose instead to use an approximation of the ideal reconstruction.

4.4.2 Cardinal Splines

The principle of the splines is to use polynomials to approximate or interpolate functions. We are interested in interpolation splines for uniform reconstruction. The cardinal splines are interpolating uniform splines widely used in computer graphics [FvDFH95a]. The s signal can be reconstructed from its samples using equation:

$$s(pT_s + t) = \sum_{i=0}^{3} c_i(t)s[p-1+i] \quad (0 \leq t < 1) \tag{4.13}$$

where the c_i functions are, for cubic splines, the following Hermite blending functions:

$$c_0(t) = \frac{1}{2}(-t + 2t^2 - t^3) \tag{4.14}$$

$$c_1(t) = \frac{1}{2}(2 - 5t^2 + 3t^3) \tag{4.15}$$

$$c_2(t) = \frac{1}{2}(t + 4t^2 - 3t^3) \tag{4.16}$$

$$c_3(t) = \frac{1}{2}(-t^2 + t^3) \tag{4.17}$$

These four polynomial functions are represented in Figure 4.16. Since the well-known Horner method can be used to perform the computation of the polynomials, the reconstruction is very efficient. This reconstruction turns out to be of very good quality too. The reason for this quality is that these Hermite functions constitute indeed a piecewise-polynomial approximation of the impulse response of a reconstruction filter, while Equation 4.13 is a small convolution between the discrete signal and this approximation. More precisely, the comparison between the Hermite functions and the (truncated) ideal *sinc* function is shown in Figure 4.17. Since the Hermite functions constitute another "Mexican hat", let us consider the windowing function that has been multiplied to the *sinc* function to obtain it. By dividing the Hermite functions by the *sinc* function, we get the functions of this "cardinal window" (see Figure 4.18). It is indeed very different from the Hann window we use in Chapter 2. But the time-domain representation is not so important. Figure 4.19 shows a comparison between the

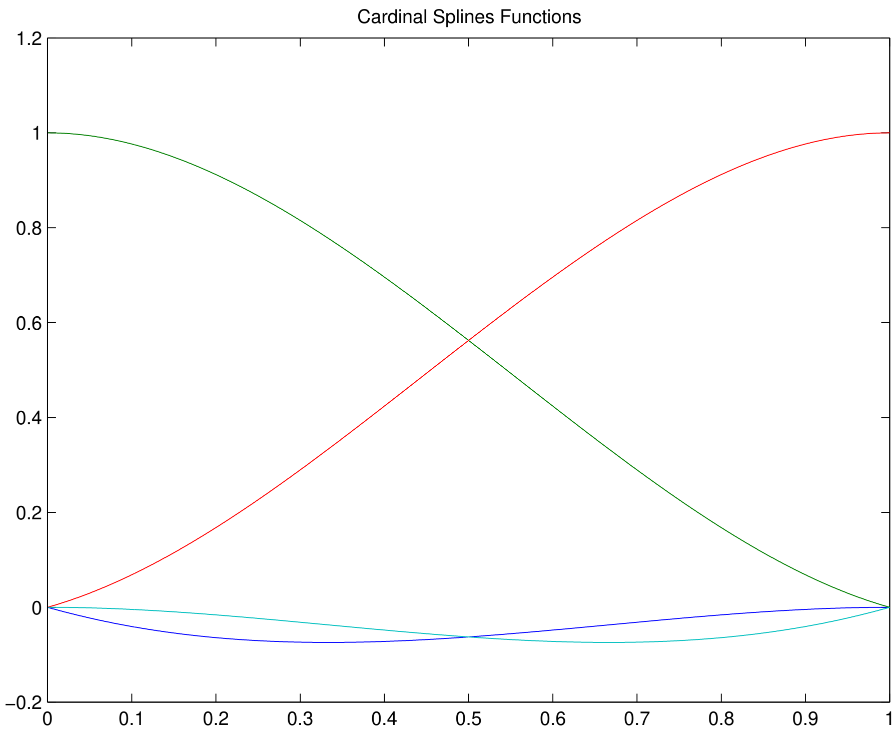


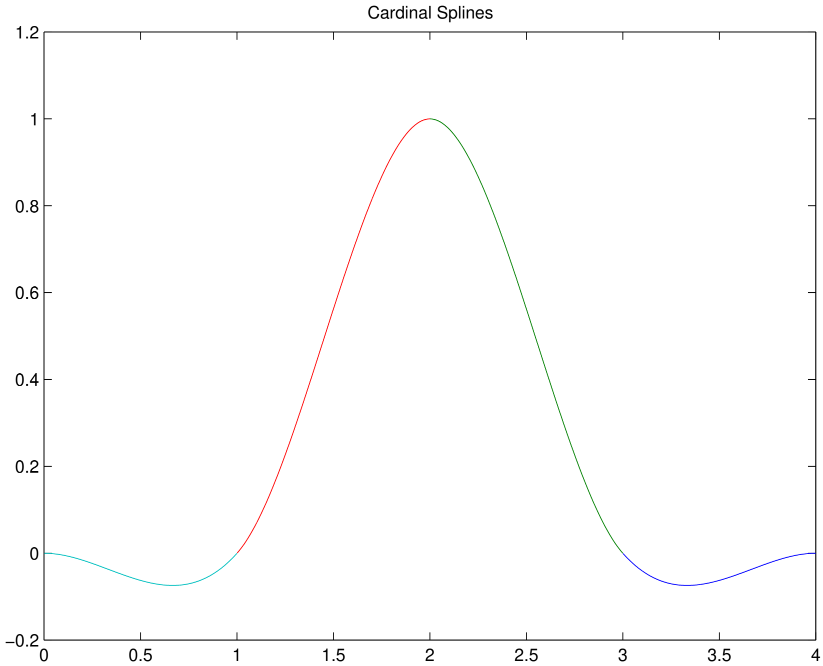
Figure 4.16: Hermite blending functions for cardinal cubic splines.

cardinal and Hann windows in the frequency domain. It appears that the cardinal window is not such a good low-pass filter in comparison to the Hann window.

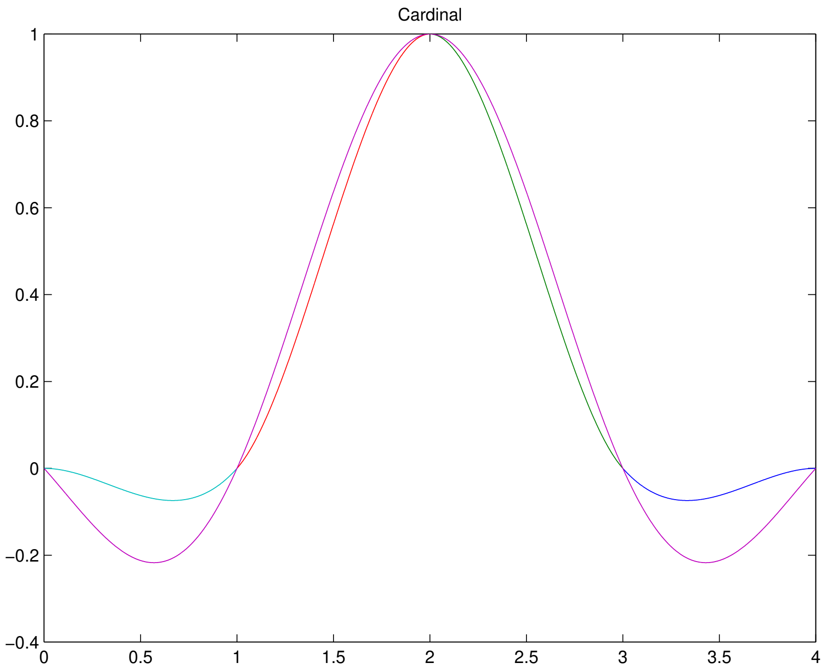
Cubic cardinal splines in conjunction to the Horner method for the evaluation of the polynomials result in a very efficient resampling technique with a sufficient precision. However we believe that it is possible to design specific splines well-suited for our purposes, and we are currently working on this.

4.5 Psychoacoustic Considerations for Synthesis Speed-Up

We have seen in Section 4.3 that we are able to synthesize hundreds – if not thousands – of sinusoidal oscillators in real time. An harmonic sound with a $F = 50$ Hz fundamental frequency requires a maximal number of 440 partials (since 22 kHz is the highest audible frequency). While we have not investigated this in details yet, we believe that a number of oscillators not much larger than that is enough for nearly all possible sounds, even polyphonic ones. The reason is that as the number of oscillators grows, the smallest distance between two oscillators is going to get smaller. When this distance is sufficiently small, either we get a psychoacoustic phenomenon known as masking [ZF81, ZF90, ZZ91], or else the two oscillators can be combined into one without altering the sound. It would then seem that there is an upper bound on the number of such oscillators that we need in order to produce most sounds. We are already very close (if not already past) this number with the method presented in Section 4.3. In fact, this number is certainly much lower, since the masking phenomenon begins when two partials of the sound lie in the same critical band (see Chapter 1). Since the whole audible spectrum can be decomposed using only 24 bands, the masking phenomenon

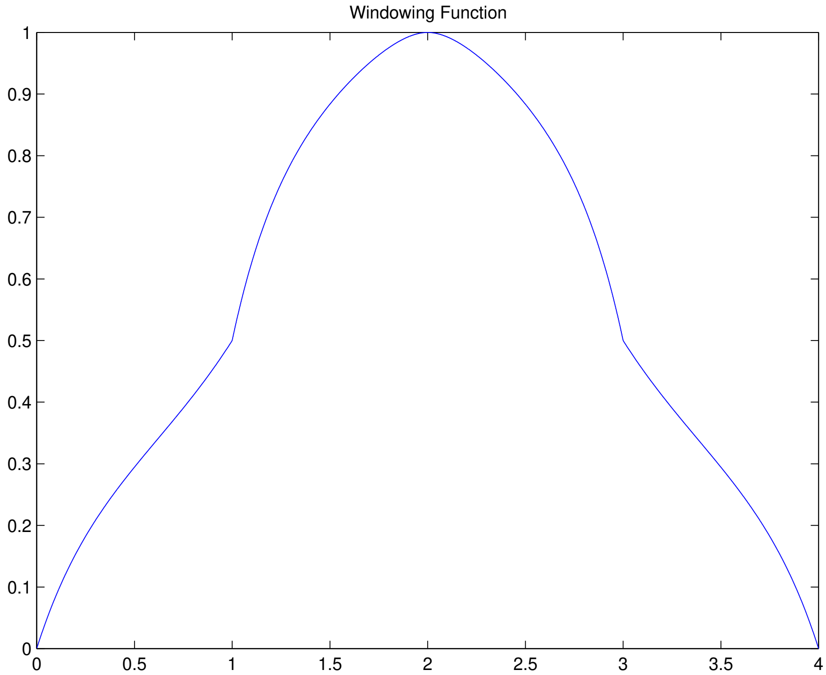


(a) Hermite functions (ordered in time)

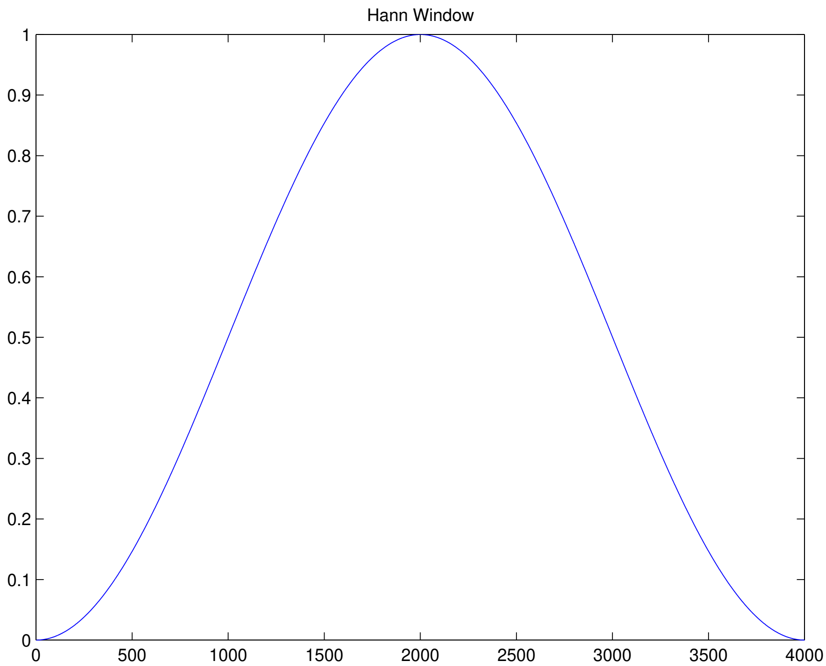


(b) Sinc function

Figure 4.17: Cardinal splines blending functions (a) versus *sinc* function (b).

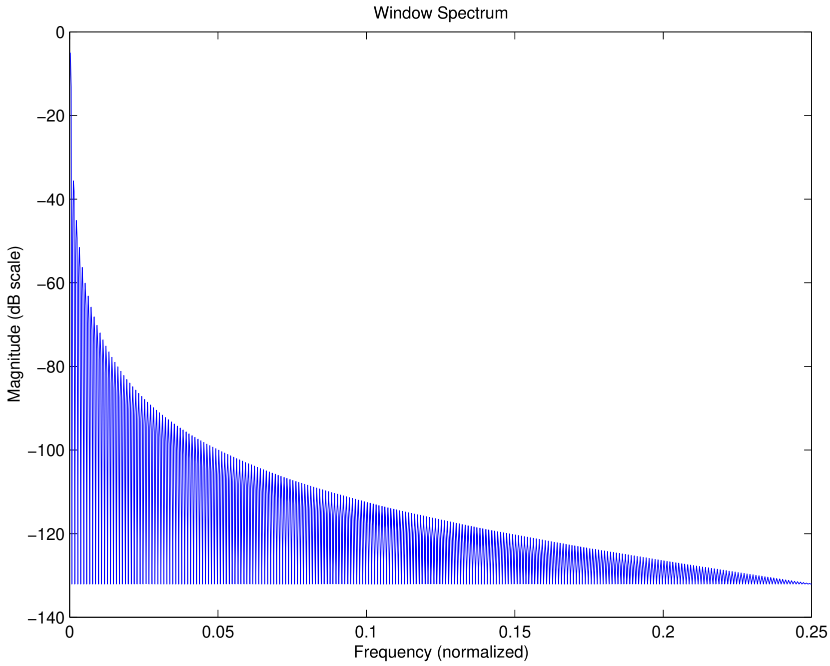


(a) Cardinal

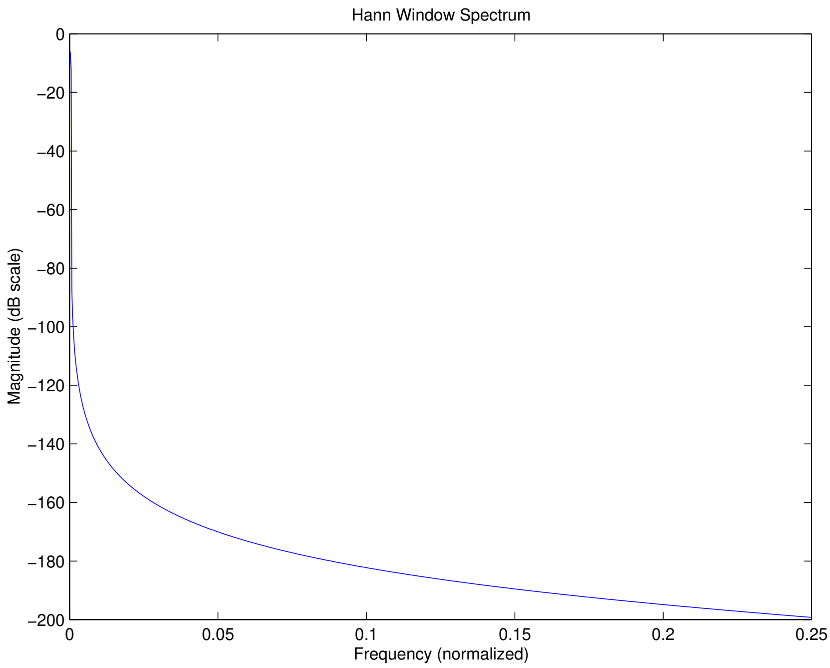


(b) Hann

Figure 4.18: Cardinal window (a) versus Hann window (b) in the time domain.



(a) Cardinal



(b) Hann

Figure 4.19: Cardinal window spectrum (a) versus Hann window spectrum (b).

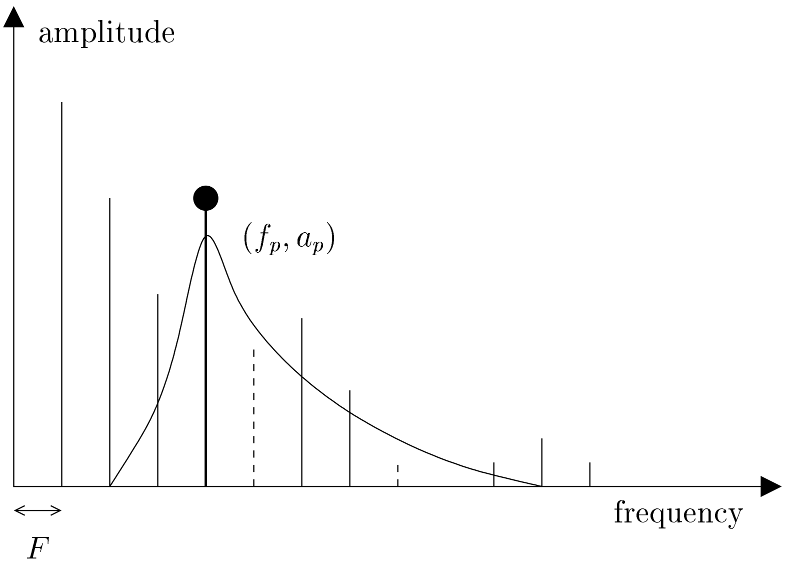


Figure 4.20: Masked partials in an harmonic sound. The two partials under the masking threshold can be removed at the synthesis stage, since they will not be heard.

appears inevitably in a sound with more than 24 partials (since then two partials are bound to lie within the same band).

4.5.1 Masking Phenomenon

Masking is a psychoacoustic phenomenon described in Chapter 1. Consider the case of two sinusoids of frequency f_M and f_m, and amplitude a_M and a_m, respectively. Assume that $a_M > a_m$. In first approximation, the masking threshold looks like a triangle in the Bark-dB scale, as shown in Figure 4.21. If f_m is close to f_M, the sound m is masked by the sound M and becomes inaudible. Garcia and Pampin use in [GP99] this masking phenomenon to reduce the number of sinusoids of a sound in the additive model. We propose to use a similar technique, but this time not for compression purposes, but for synthesis efficiency only. The masking phenomenon is also used in the MPEG-II Layer 3 audio compression [BB95, Pan95, HBEG95]. We do not want to compress the spectral sounds, because we need all the partials – even the inaudible ones – for the musical transformations (see Chapter 2). But masked partials can be removed at the synthesis stage, since they will not be heard.

Garcia and Pampin use a simple masking model to evaluate the signal-to-mask ratio (SMR) of each partial. This model consists of:

- The difference Δ between the level of the masker and the masking threshold (-10 dB);
- The masking curve towards lower frequencies (left slope: 27 dB/Bark);
- The masking curve towards higher frequencies (right slope: -15 dB/Bark).

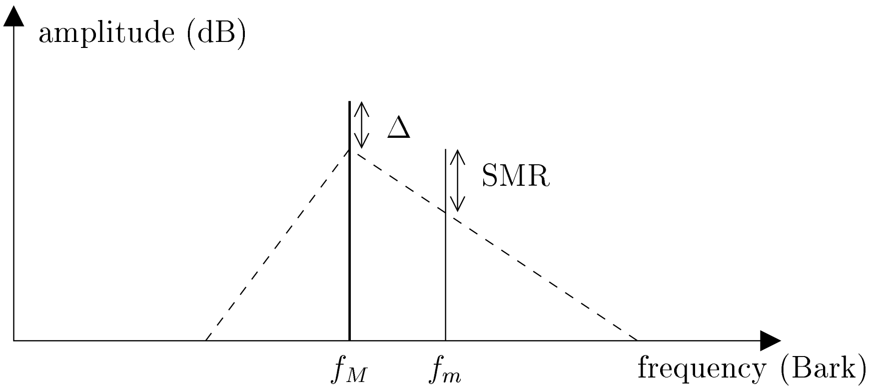


Figure 4.21: Masking of a sinusoid of frequency f_m by another sinusoid of frequency f_M. The masking effect is maximal when f_m and f_M are close. As a first approximation we can consider that the masking threshold is close to a triangle in the Bark-dB scale, although it is not exactly the case in practice, especially for the top of the triangle.

4.5.2 Enhanced Algorithm

Before the synthesis algorithm itself, another algorithm can be added in order to reduce the number of partials to be synthesized. This algorithm is very simple. It consists in scanning the partials, sorted by decreasing amplitudes. Then, for each partial we compute the signal-to-mask ratio SMR for each of its neighbors. If SMR < 0 then we remove the neighbor from the set of partials.

From the complexity point of view, this algorithm can be up to quadratic if no partials are masked. In such a case, for every partial the whole set of partials is scanned – for nothing – and the worst complexity is $O(P^2)$, where P is the number of partials.

Fortunately, this algorithm is equivalent to another one, which has a linear complexity. The idea is to construct a global masking level, uniformly sampled in the Bark scale. At the beginning of the algorithm, this masking level is set to 0 ($-\infty$ dB). All the partials are then scanned by decreasing amplitudes. For each partial, if its level is above the masking level corresponding to its frequency then it is removed from the set of partials. If it is not masked, then its own masking level is accumulated into the global masking level (the maximum values in the two levels are considered).

This algorithm requires the sampling of the global masking level, with a sufficient number of samples to ensure a good precision. The problem is that this computation must be worth doing in a real-time synthesis algorithm. It must take less time eliminating the useless (inaudible) partials than synthesizing them anyway. We are currently working on making this algorithm very efficient.

Further research includes exact determination of other psychoacoustic phenomena such as temporal masking and other related ones that may help us decrease the number of oscillators needed, and thereby increasing performance.

As a consequence, in a few years most people will have a PC with sufficient power on their desks to generate any sound in real time based on our method.

4.6 Structured Additive Synthesis

The synthesis of sounds in the Structured Additive Synthesis (SAS) model (see Chapter 2) consists in first reconstructing the partials of the classic additive synthesis model from the structured parameters, then synthesizing them using the method described above.

4.6.1 Computation of the Additive Parameters

The computation of the additive parameters is done periodically. We currently use a rate of about 172 samples per second to match *ReSpect* requirements. But this will be much lower in the near future, since the SAS parameters are band-limited to a frequency much lower than $172/2 = 86$ Hz. In fact they are band-limited to only 20 Hz.

Let us consider a sound in the SAS model: $S = (A, F, C, W)$. The number of partials to generate at time t is:

$$P(t) = \left\lfloor \frac{F_{\text{max}}}{F(t)} \right\rfloor \tag{4.18}$$

where F_{max} is the highest audible frequency (about 22 kHz). Then, for each of the P partials, the additive parameters have to be calculated.

Frequencies

The frequency of the p-th partial is given by the following equation:

$$f_p(t) = W(pF(t), t) \tag{4.19}$$

Amplitudes

Of course we have:

$$a_p(t) = A(t)\ C(f_p(t), t)$$

provided that C is normalized, that is:

$$\sum_{p=1}^{P} C(f_p(t), t) = 1$$

In fact this normalization can only be done on the fly at the resynthesis stage, and we have:

$$a_p(t) = \begin{cases} A(t)\ C_{\text{norm}}(f_p(t), t) \\ A(t) \frac{C(f_p(t),t)}{\sum_{p=1}^{P} C(f_p(t),t)} \\ \sqrt{2} A_{\text{RMS}}(t) \frac{C(f_p(t),t)}{\sqrt{\sum_{p=1}^{P} (C(f_p(t),t))^2}} \end{cases} \tag{4.20}$$

depending on whether or not we are considering the RMS amplitude A_{RMS} (see Chapter 2).

4.6.2 Simultaneous SAS Sources

A sound in the SAS model is always a monophonic source. The problem is now to be able to play polyphonic sounds, that is sets of monophonic sources. Each SAS source maintains a set of active partials. The size of this set is given by Equation 4.18. At each synthesis frame, the instantaneous frequency and amplitude of each partial are recomputed using Equations 4.19 and 4.20. Of course the number of active partials may vary over time. If this number increases, then new partials are born (and must be added at the end of the stream of partials according to the protocol of *ReSpect*). If this number decreases, then some partials die.

The SAS software library was specially designed for the manipulation of sounds in the SAS model. In the SAS library, there is a very tricky and efficient algorithm to interleave the partials of the sources into a single stream respecting the protocol imposed by *ReSpect*. Its complexity is a linear function of the number of partials.

Roughly speaking, the synthesis engine maintains an ordered set – a list – of references to the active partials among the different SAS sources. The trick is that each partials contains a reference to its entry in this list. When a partial is born, a new entry is allocated for it at the end of the engine list. When a partial dies, its entry is set to the `NULL` value, and then got recycled at the next synthesis frame. Recycled means that the cell in the list corresponding to this hole is removed when the list is browsed for the computation of the *ReSpect* frame. The engine list is in fact implemented inside an array in a very efficient way.

4.6.3 SAS Library Overview

The SAS library was specially designed for the manipulation of sounds in the SAS model.

SAS Frames

The SAS sounds consist of a sequence of frames, which are the result of the uniform sampling in time of the SAS parameters. The `sas_frame` type is available in the SAS library:

```
typedef
struct sas_frame
{
  sas_value    A;
  sas_value    F;
  sas_envelope C;
  sas_envelope W;
}
*sas_frame;
```

The envelopes are vectors resulting from the uniform sampling in frequency of functions. The library provides basic functions for their manipulation, such as resampling operations or the reconstruction of the associated continuous-frequency functions.

SAS Sources

The SAS sounds are represented by the `sas_sound` type. Basic operations on these sounds are available, such as:

- Creating and deleting sounds:

```
sas_sound sas_sound_make (double rate, int size);
void      sas_sound_free (sas_sound sound);
```

- Gathering information:

```
double sas_sound_rate (sas_sound s);
int    sas_sound_size (sas_sound s);
```

- Getting / Setting frames inside sounds:

```
sas_frame sas_sound_get (sas_sound s, int i);
void      sas_sound_set (sas_sound s, int i, sas_frame f);
```

- Loading or saving a sound:

```
sas_sound sas_load (char *name);
int       sas_save (sas_sound sound, char *name);
```

SAS Synthesis
The following functions allow the user to enter or leave the SAS synthesis engine, respectively:

```
bool sas_enter (void);
void sas_leave (void);
```

The value returned by `sas_enter()` indicates whether or not the initialization of the engine was successful.

For the synthesis itself, the library uses synthesis handlers (`sas_handler` type) which are handlers to active SAS sources, that is sounds that are currently played. There are functions to create or delete a handler:

```
sas_handler sas_create (void);
void        sas_delete (sas_handler source);
```

and an update function to refresh the parameters of each source at each synthesis frame:

```
int sas_update (sas_handler source, sas_frame frame);
```

Once all the sources have been updated, a synthesis function is called:

```
int sas_player (void);
```

The following program plays two SAS sounds simultaneously:

```
#include <stdlib.h>
#include <stdio.h>
#include <sas.h>

#define MIN(_X_,_Y_) ((_Y_)<(_X_)?(_Y_):(_X_))

int
```

```
main (void)
{
  int i, n;
  sas_sound s1, s2;
  sas_handler h1, h2;

  /* load sources */

  s1 = sas_load ("source1.sas");
  s2 = sas_load ("source2.sas");

  /* enter SAS engine */

  if (!sas_enter())
    {
      fprintf (stderr, "can't open spectral device!\n");
      exit (1);
    }

  /* create synthesis handlers */

  h1 = sas_create ();
  h2 = sas_create ();

  n = MIN (sas_sound_size(s1), sas_sound_size(s2));

  /* play n frames */

  for (i=0; i<n; i++)
    {
      sas_update (h1, sas_sound_get (s1, i));
      sas_update (h2, sas_sound_get (s2, i));

      sas_player ();
    }

  /* delete handlers */

  sas_delete (h1);
  sas_delete (h2);

  /* leave SAS engine */

  sas_leave ();

  /* free sources */
```

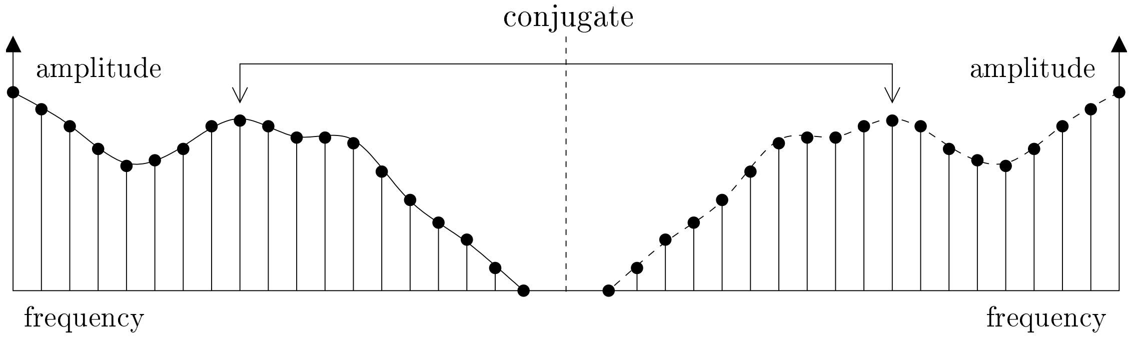


Figure 4.22: The discrete spectrum is reconstructed from the uniform sampling of the noise envelope (bins 1 to $N-2$, since bins 0 and $N-1$ correspond to the DC component, always set to 0).

```
  sas_sound_free (s1);
  sas_sound_free (s2);

  /* the end */

  exit (0);
}
```

4.7 Noise Synthesis

The SAS model can be extended to include noise (see Chapter 2). However, the synthesis method for noises is very different, since they are not made of partials. The method for noise synthesis described in this section is similar to the one used in SMS [Ser89, Ser97b]. The only difference is that the spectral envelope is not a piecewise-linear approximation as in SMS, but the continuous color C parameter of the SAS model.

4.7.1 Spectrum Approximation

The spectrum of a noise can be approximated by a spectrum with peaks very close in frequency [ZF81]. When the distance between two adjacent peaks does not exceed 1% (10 Hz around 1000 Hz), this is indeed a very good approximation. As a consequence, with a sufficient number of bins, the inverse Fourier transform can synthesize noise. The idea is to set the value of the DC component (bin 0 and $N-1$) to 0, then for the other bins (AC components) to use amplitudes given by the spectral envelope (color C in the SAS model). The phases for all bins are chosen randomly at each synthesis frame. In fact only the first half of the spectrum has to be reconstructed, since a necessary condition for the resulting signal to be real-valued is that the other half must be its conjugate symmetric (see Chapter 1). Figure 4.22 illustrates the reconstruction of the spectrum from the spectral envelope.

4.7.2 Inverse FFT

Once the spectrum has been approximated, the Inverse Fast Fourier Transform (IFFT) is applied in the same way as in the Inverse FFT method of Section 4.1.

4.7.3 Overlap-Add Technique

The same overlap-add technique as in Section 4.1 is then used to avoid clicks. Although the Bartlett (triangular) window is often used in practice, we use the Hann window since it reduces some of the synthesis artifacts produced by the overlap-add synthesis, thus leading to a better perceptive quality.

Chapter 5

Applications and Perspectives

Applications of the SAS model (see Chapter 2) are numerous in the fields of computer music for creation and education. This chapter presents some of these in-progress applications, some of them being in a very early stage while already providing us with promising research topics for the near future.

Section 5.1 presents some of the applications of the SAS model to sound exploration and design. One of the advantages of this model is its aptitude for creating hybrid sounds from the combination of several sounds. Section 5.2 shows the use of the SAS model for musical composition. Since this model favors the unification between music and sound at a sub-symbolic level [Lem93], it enables composers to modify both the micro-structure and the macro-structure of musical pieces in a multi-scale composition and thus to perform musical compositions on several time scales in a continuous manner. A new sound synthesis language for musical composition has been implemented and should provide a way to validate and enrich the model. Section 5.3 deals with the use of the SAS model for interactive control. A pedagogical tool for early-learning electro-acoustic music is based on this model. It provides sound controls that are well-suited for young children because they are based on sound listening rather than signal synthesis. This tool can also be used for real-time musical performances, and it is in fact the first step to a new visual language for music. Finally, Section 5.4 introduces in-progress applications of the SAS model to singing voice. This model is indeed well-suited for the representation of vowels and allows us to control in time the volume and the pitch as well as the timbre itself.

5.1 Sound Design

Concrete music [Sch94] is traditionally composed using sampled sounds manipulated with sound editors based on the simple cut-and-paste operation. The representation of sound as a time signal offers no precise control of the musical parameters such as the volume, pitch or duration. Moreover, as mentioned in Chapter 2, these parameters are interdependent in the temporal representation of sound.

Using the SAS model, we propose to leave the collage art for the sculpture of sound. The SAS model constitutes a solid base for investigating scientific and musical research on the notion of timbre. This model is of great interest for exploring sounds and creating hybrids.

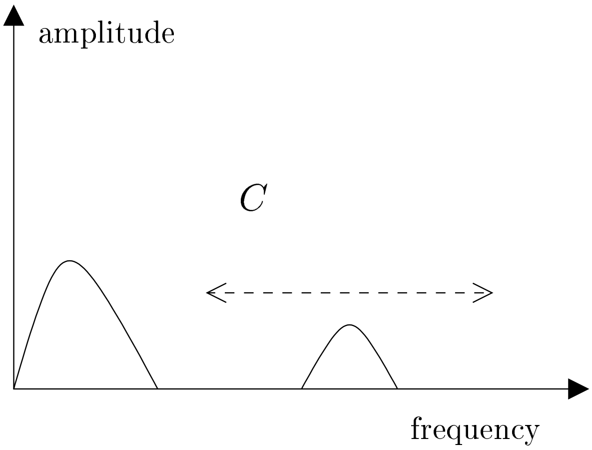


Figure 5.1: The didjeridoo shows characteristic variations of the color parameter C. A formant is nearly constant in the low frequencies, while another one keeps on moving up and down across the higher frequencies.

5.1.1 Exploring Sounds

Since there is a close correspondence between the parameters of the SAS model and real music perception, it is possible to analyze sounds and inspect their inner structures from an educational standpoint for example. It is of course possible to eliminate analysis entirely and create new sounds directly using the parameters of the model. For example, the didjeridoo is an Aboriginal musical instrument which shows characteristic variations of the color parameter, as illustrated in Figure 5.1. It is rather simple, in the SAS model, to design a sound perceived as it had been produced by a real didjeridoo. In order to manipulate the two-dimensional parameters of the SAS model (color C and warping W), grey-scale pictures can be used to represent the parameter value as a function of both time and frequency (see Figure 5.2). This way of editing these two-dimensional parameters has turned out to be very convenient, even if a three-dimensional plot of the associated surface – the parameter value being the elevation of the surface – is also useful since the eye is less sensitive than the ear. Those – often very large – surfaces can be rendered using specific algorithms. Many of them can be found for example in [FvDFH95b].

5.1.2 Using Hybrid Sounds

As mentioned in Chapter 2, one of the advantages of the SAS model is its aptitude for creating hybrid sounds from the combination of several sounds. Regarding sound hybridization, our *ProSpect* software tool opens new horizons for both researchers and composers. The SAS model permits to consider sound as a musical material well-suited for "musical sculpture" using curves [Arf98], and not as "black boxes" on which abstract functions are applied. We are developing a sound synthesis language based on SAS – part of our *ProSpect* software package – in close collaboration with composers of electro-acoustic music. This language has been used directly – that is without additional software tool or GUI – during the compositional process of the fourth fragment of the piece "Sept Couronnes pour Goethe" [Riv99] by Rivet in 1999. More precisely, hybrid sounds were used in this piece either to draw a link between different

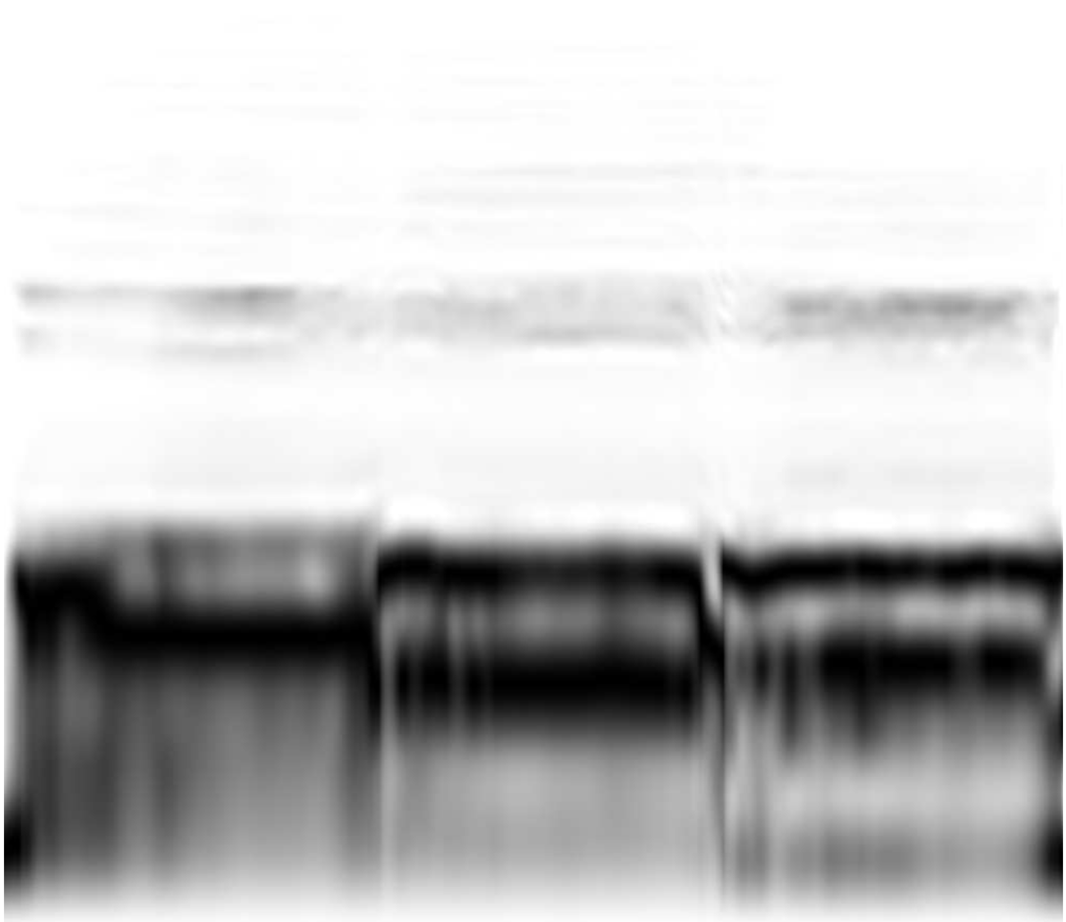


Figure 5.2: Representation of the SAS color parameter as a grey-scale level as a function of both time (horizontal axis) and frequency (vertical axis). The lowest (zero) and highest levels are displayed as white and black, respectively. The corresponding sound is the *a* vowel sung over three notes. We can clearly see the evolutions of the color with pitch, and more precisely the three notes of different pitches as well as the two short transition areas between these three notes.

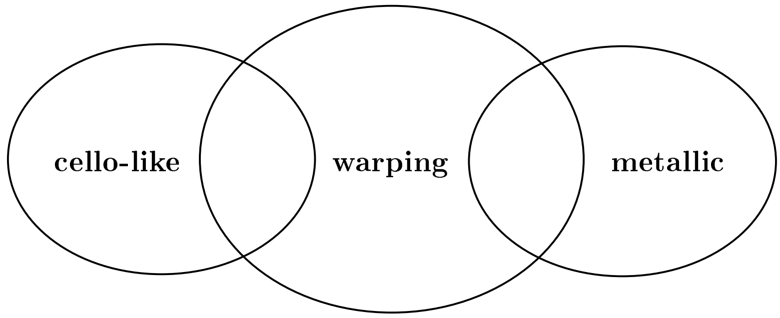


Figure 5.3: Using warping to make a family of hybrid sounds, to go from a family of sounds to another one.

families of sounds or to renew listening within a certain family of sounds. These hybrid sounds turned out to be extremely useful during the compositional process.

5.2 Musical Composition

Since the very beginning of the research in the field of computer music [Moo90, Roa96], two main trends are developing simultaneously but separately. The first trend deals with symbolic musical structures and is based on the intentions of the composer in order to allow always more sophisticated musical abstractions. This research on symbolic modeling of music is issued from the work of Hiller [HI59] on automatic composition. Musical pieces are defined using atoms (notes) and musical structures represented using various paradigms such as object-oriented, functional or constraint-based programming. The organization in time of these musical structures [PC98] is also studied in details. Software tools in this symbolic domain are numerous and quite powerful. However their power comes up against the boundary separating the note from the sound, since they cannot penetrate the opaque sounds – considered as “black boxes” – in order to efficiently control the micro-structure of the symbolic atoms. The second trend considers computers as instruments for musical sound synthesis [Pie83] rather than tools for assistance for musical composition. While the research related to music mainly concerns musical analysis and composition, the research related to sound deals with sound modeling, analysis, synthesis, and transformation in order to directly manipulate the inner structures of sound. This research on the sound structure is issued from the work of Risset and Mathews on the analysis of musical instrument tones [RM69].

The families of software programs in the field of assistance for musical composition follow this sound / music dichotomy. The composers have to manipulate different kinds of software tools in order to compose musical pieces and they must entirely manage every interaction between the sounds and the musical macro-structures within their pieces. This is done at the expense of uncomfortable switches from one representation to the other, indeed even from one software tool to the other. Even in many languages for musical sound synthesis [BPS97] like CSound [Ver86, Ver92] or MPEG-4 Structured Audio [VGS98], music and sound are clearly separated into score and instrument files, even if the arbitrary boundary between those files is far from being obvious from the programmer's point of view…

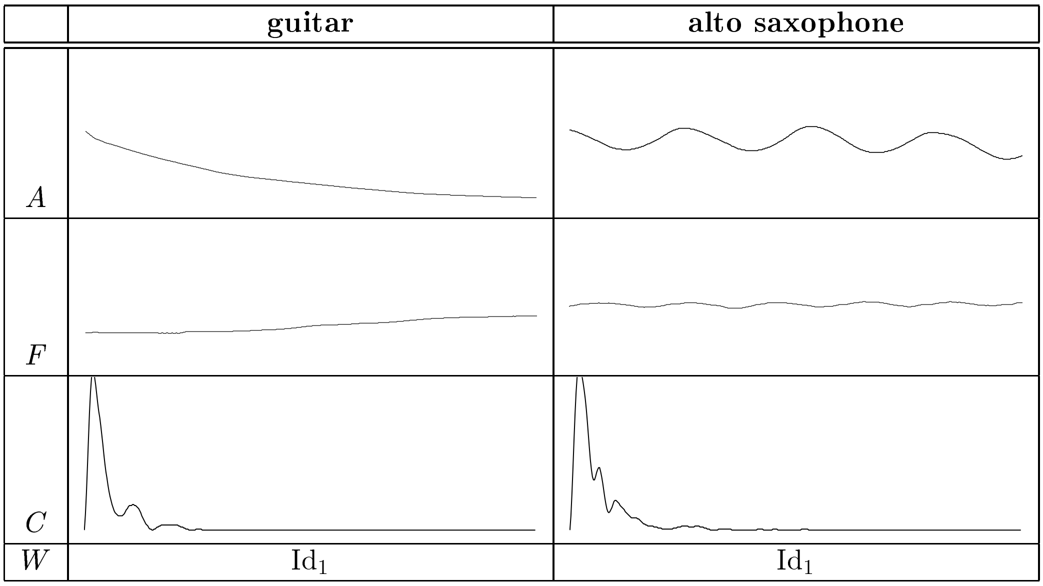


Figure 5.4: Examples of instrumental sounds in the SAS model. From top to bottom are displayed the four parameters of the SAS model: the amplitude and frequency as functions of time $A(t)$ and $F(t)$, the color as a function of frequency only $C(f)$, since it does not vary much over time for the examples considered here, and finally the warping which satisfies $W(f,t) = f$ because both sounds are harmonic.

5.2.1 Unifying Sound and Music

We propose to focus on the perception of the sound rather than its physical cause, in order to unify sound (microscopic) and music (macroscopic). We propose as well to consider the musical intentions of the instrumentalists instead of their physical actions on the instruments.

We also propose to try unifying musical writing and sound control in order to permit musical composition on several time scales in a continuous manner [DCM99b, DCM99a]. The SAS model was specially designed for this purpose. It is based on parameters close to both perception and musical terminology, and permits a correspondence between microscopic (sound) and macroscopic (music) structures. We have experienced that this unification enables migrations of the control among the different levels, from the microscopic (sound) to the macroscopic (music) one, thus enriching the palette of the composer.

The ear is an arithmetic analyzer while the human mind is a symbol manipulator. But the sub-symbolic representation assumes that there is more to mental representation than just symbols. For example, when we imagine a chord we do not recall the symbols, but we hear it internally. By defining the sound as the temporal evolutions of parameters close to both perception and musical terminology, the SAS model favors the unification between music and sound at a sub-symbolic level.

The SAS parameters are closely related to the musical ones. Figure 5.4 shows the results produced by *InSpect* (see Chapter 3) on a guitar and an alto saxophone. Regarding the guitar, one can observe a fast decreasing of the amplitude. The small increase of the frequency (portamento) at the end is due to a pull performed on the string by the instrumentalist. Regarding the saxophone, one can clearly see a tremolo (sinusoidal amplitude variation) as

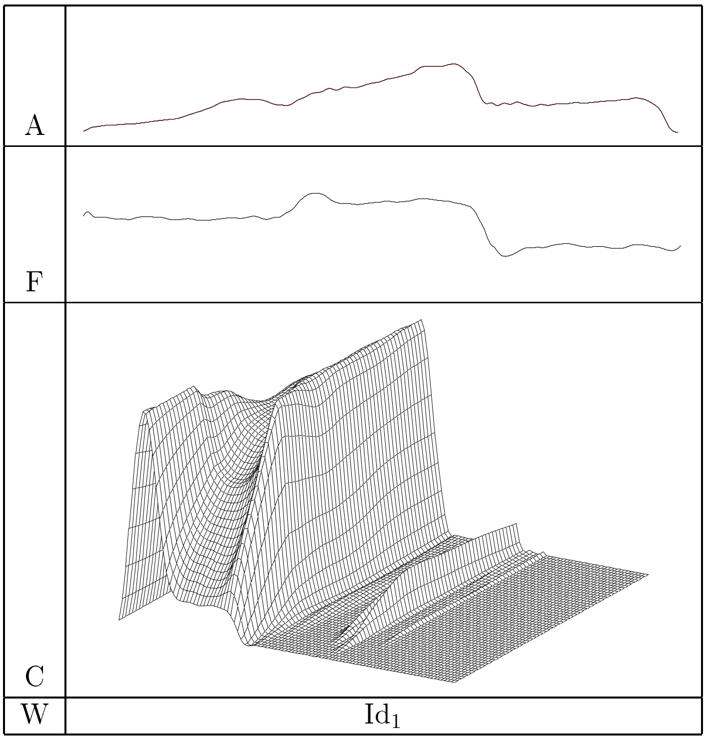


Figure 5.5: Singing voice in the SAS model. The corresponding sound is the same *a* vowel sung over three notes as in Figure 5.2. The color parameter is displayed here until the half of the first note only.

well as a small vibrato (sinusoidal frequency variation). It appears on these examples that the color is nearly constant over time and that these instruments are perfectly harmonic (thus $W = \mathrm{Id}_1$). Figure 5.13 shows possible colors for French vowels, and Figure 5.5 represents the *a* vowel sung over three notes. One can note, then, that the distinction between sound and musical parameters is not clear anymore. Indeed, considering the examples of Figure 5.4, the portamento done by the guitarist is both a sound and musical parameter. The same remark can be done for the tremolo and the vibrato of the saxophone. Regarding the singing voice example illustrated in Figure 5.5, one can clearly read the dynamic and the melody of the song in, respectively, the A and F parameters of the sound. The arbitrary distinction between music and sound parameters tends to simply disappear.

The SAS model is intended for composers willing to write pieces while using the possibility to modify the inner structures of the sounds, using the same vocabulary. The rate of the modification of the model parameters determines the musical scale we are modifying, ranging from the musical macro-structures – traditionally notated – to the control of the timbre itself [Vag94], and this in a continuous manner, that is without encountering a boundary between music and sound.

We consider only monophonic sound sources, which should be spatialized [Bla97]. By the way, the position in space is a parameter which should be as musical as physical. The parameters of the SAS sound model are very close to the musical parameters and macro-structures. Most musical sound transformations can be expressed as more or less rapid variations of the

Tempo	Amplitude	Frequency	Color	Warping
Writing (0-≈8 Hz)	Dynamic, Crescendo	Melody, Trill	Orchestral Sonority	Chords, Aggregates
Control (≈8-20 Hz)	Tremolo, Roughness	Vibrato, Scintillating	Spectral Envelope	Spectral Harmonicity
Hearing (20-22 kHz)	listening only			

Figure 5.6: Some relations between musical terminology and the four SAS parameters, for different ranges of variation rates.

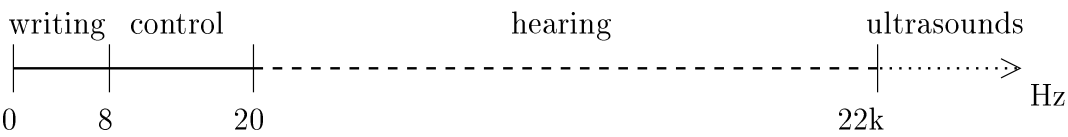


Figure 5.7: Frequency scale for the rate of the parameter variations.

model parameters. We show the proximity between sound parameters, musical ones, and musical macro-structures, as well as their use for multi-scale composition.

Multi-Scale Composition

Most musical transformations can be simply expressed as SAS parameter variations. Depending on the rate of these variations, composers can modify both the micro-structure and the macro-structure of musical pieces in a multi-scale composition [Vag98, DCM99b, DCM99a]. When the variations are slow enough, they can be written on a score. This is the domain of writing. When they are too fast to be written, we enter the domain of control (or interpretation). Figure 5.6 gives a brief summary of the relations between musical terminology and SAS for these two domains.

Time, Tempo, and Rhythm

The frequency domain can be schematically split into four main areas, as shown in Figure 5.7. A similar division has already be done by Vaggione [Vag96, Vag98]. Above 22 kHz are the ultrasounds, whose frequencies cannot be perceived by the human auditory system. Our auditory area is located approximatively between 20 Hz and 22 kHz. These frequencies are too fast to be precisely controlled by human beings. The SAS parameters are functions of time and can therefore be considered as control signals. In no way they could vary with a frequency much greater than 20 Hz, since in this case their variations might be audible, thus disrupting the inner structures of the sounds: This is the modulation phenomenon, basis of famous nonlinear synthesis techniques [Moo85b, Cho73, Arf79], but undesirable here since it questions the perceptive consistency of the parameters.

Consequently all the controls made within the SAS model will be below the threshold of hearing, thus being “slow time-varying”. More precisely, we will consider them as band-limited to a frequency below 20 Hz. Below 8 Hz, corresponding to a tempo of about 500 beats per minute, it is possible to notate the parameter variations in a concise way. This is the area of writing, widely used in the compositional process. Above 8 Hz, one enters the area of control, interpretation, and "effects".

The SAS model covers both the control and writing domains: From sound (microscopic scale) to music (macroscopic scale) it is only a matter of speed of variation for the perceptive parameters.

Moreover the control of time and its associated macro-structure – rhythm – can be done very easily as a time-stretching like this: $t' = k(t)\ t$. We speak about temporal expansion or compression for k, respectively, lower or greater than 1.

Amplitude, Loudness, and Dynamic

We have already mentioned the close link between amplitude and loudness in Section 2. The fastest amplitude variations contribute to the "roughness" of the sound, whereas slower variations constitute the dynamic of the music. Indeed the crescendo or decrescendo reflect, respectively, the increase or decreasing of the loudness, and thus of the amplitude. One can easily modify the amplitude of a sound using a simple formula such as: $A'(t) = k(t)\ A(t)$. If k is a constant, we get a simple amplification. If k is a sinusoid with a frequency around 10 Hz, the musical effect obtained is a tremolo with the corresponding frequency. If the variations of k are slow and monotonous (mathematically speaking), we get a fade-in or a fade-out for, respectively, an increasing or a decreasing function. The terms amplification, tremolo, fade-in and fade-out, as well as many others deal with more or less rapid variations of the amplitude. This is a matter of scale in the rate of variation of the A parameter.

Frequency, Pitch, and Melody

There is also a close link between frequency and pitch. The fastest frequency variations contribute to the "scintillating" sound sensation, whereas slower variations constitute the melody of the music. One can easily modify the frequency of a sound using a simple formula such as: $F'(t) = k(t)\ F(t)$. If k is a constant, we get a simple transposition. If k is a sinusoid with frequency around 10 Hz, the musical obtained is a vibrato with the corresponding frequency. If the variations of k are slow and monotonous (again mathematically speaking), we get a glissando or a portamento. If these variations are not monotonous – and quite fast – then we obtain a trill instead. The terms transposition, vibrato, glissando, portamento, trill, as well as many others deal with more or less rapid variations of the frequency. Again, this is only a matter of scale in the rate of variation of the F parameter.

Color and Timbre

Although loudness and pitch are the most common parameters in the occidental music – indeed the only parameters (apart from duration) represented in the traditional music notation – we must not forget the color, which is essential for oriental musics. For harmonic sounds, the variations over time of the color completely constitute the "timbre", whose definition is probably the most difficult but also the most interesting. From a musical point of view, one can note the importance of the orchestral sonority, each instrument having its own colors which blend more or less harmoniously with the others. One can also speak about the coloration induced by broadcasting systems or concert halls for example. The modification of the color (coloration) can be done like this: $C'(f,t) = \mathcal{F}(f,t)\ C(f,t)$. If $\mathcal{F}$ is constant over time then we get classic filtering ($\mathcal{F}$ is the frequency response – the color – of the filter). If $\mathcal{F}$ varies over time, we can create very interesting hybrid sounds.

Warping and Harmonicity

Harmonicity plays a key role in musical harmony [MP80a, MP80b]. For example a nearly-harmonic chord gives a sensation of frequency, but also of inharmonicity. These sensations are related to the F and W parameters of the resulting sound. In order to model such a chord, one can for example organize the model in a hierarchy by extending it with symbolic structures such as the union. A chord (polyphony) of N sources can then be expressed like this:

$$\bigcup_{i=1}^{N} \{(A_i, F_i, C_i, W_i)\}$$

For a nearly-harmonic chord, it is possible to get A, F, C, and W parameters resulting from those of the different sources, by considering for example the N (F_i, A_i) pairs and by using the same structuring methods as the ones described in Chapter 3. Harmonicity is probably the least-known of the four SAS parameters, and would certainly deserve a full study.

5.2.2 *BOXES* Software Package

BOXES [Beu00, MB00] is a digital audio sequencer with constraints developed by Beurivé. It is an experimental software tool in the field of assistance for musical composition, dealing with spectral models available in *ProSpect* and thus inheriting from *ProSpect* the possibility for advanced sound transformations as well as real-time synthesis. Besides the possibility to manipulate spectral sounds, *BOXES* innovates in the field of assistance for musical composition by proposing an original mechanism for edition based on hierarchical structures together with interactive updates using constraints, which can greatly facilitate the compositional process. Constraints are also successfully used in MusicSpace [PD99] by Pachet and Delerue for the control of music spatialization.

In *BOXES* the graphical representation of a spectral sound consists of two time points (start, end), a series of amplitude values (corresponding to the A parameter of the SAS model), and a series of frequency values (corresponding to the F parameter of the SAS model). Operations on sound allow to change time points (onset and duration), amplitude, and frequency. The sounds within a musical piece are organized in a hierarchical structure (representing chords, melodies, groups, tracks, etc.), as shown in Figures 5.8 and 5.9. Music composition is performed with high-level structures (not necessary tracks) and manipulations. The hierarchical structure can be updated interactively by the mean of constraints placed on time points, amplitudes or frequencies of the sounds within the structure. However the synthesis process does not immediately take these updates into account yet. Among the constraints available for interactive updates we can cite temporal constraints (such as “before”, “overlap”, etc.) or proportionality constraints (proportional amplitudes, proportional frequencies, etc.) for example.

5.3 Interactive Control

While music composition could be done in a non-interactive way, musical performances or virtual instruments require an interactive control of sound.

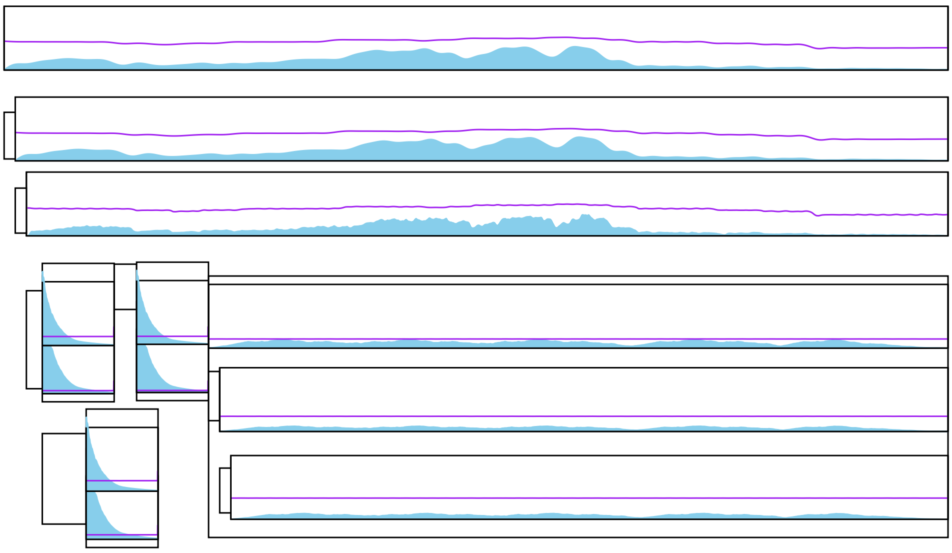


Figure 5.8: A musical piece composed using *BOXES*.

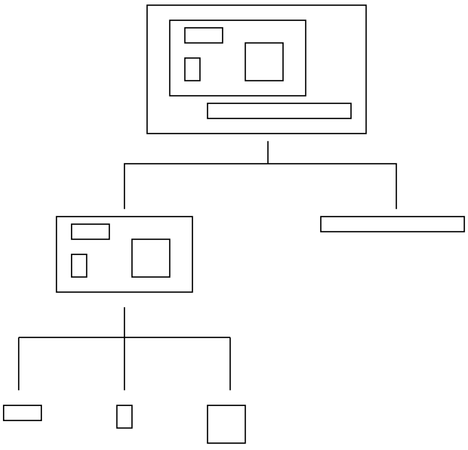


Figure 5.9: A hierarchical structure (top) and its decomposition in several sub-structures, which can in turn be decomposed, and so on.

5.3.1 Visual Language for Music

Sound synthesis often consists in implementing a sound generator using a language derived from MUSIC V for the definition of the synthesis algorithms. In this case the parameters manipulated by the composer are interconnections between modules such as oscillators, adders, multipliers, filters, etc. The synthesis possibilities are numerous, but the parameters manipulated by the composer are only remotely related to the musical parameters as perceived by a listener.

An interesting research field is the definition of a visual language for music. We are thinking about a structure consisting of interconnected processing nodes with flows of sound parameters between two adjacent nodes. This structure is already well-known since it is present in musical software architectures such as Kyma [Sca89, Sca87], Max [Tod96] or jMax [DCMS99, Déc00] for example. However we propose to consider flows of SAS parameters instead of flows of sound samples in the temporal model. Since the manipulations are then made on a stream of spectral parameters with a low rate, they are indeed quite efficient. Another important point is that the flows of control parameters can also be expressed in the SAS model, thus eliminating the need for other kinds of data flows (such as MIDI events). This visual language already exists as a tool for the purposes of education called *Dolabip*.

5.3.2 *Dolabip* Project

Since the SAS model is based on perceptive and musical criteria, we think it is well-suited for computer-assisted early-learning music. *Dolabip* is a multi-field project whose objective is the creation of a meta-instrument to be used for early-learning electro-acoustic music. Practical experience in nursery school is lead by a musician, a teacher, a psychologist, and a music teaching specialist.

This project is composed mainly of two parts (see Figure 5.10). The first one is an hardware device consisting of potentiometers and buttons well-suited for manipulation by children. The second one is a software tool producing sounds according to the data sent by the device. The software tool allows the user to change the way the data get interpreted. The development teams of the software and hardware parts build tools that are necessary for experimenting the pedagogical program.

The software tool relies on an tree-like structure composed of interconnected processing nodes. Flows of SAS parameters travel between adjacent nodes. This structure represents the expression computing the sound from the data either statically stored in memory or dynamically sent in real time by the external control devices (manipulated by the children). The SAS parameters produced by the root of the tree constitute a sound in the SAS model, which is then synthesized in real time. This real-time synthesis is performed using *ReSpect* (see Chapter 4), which guarantees that there will be no click during the live performance: In the worst case the sound will remain steady for a few milliseconds if the spectral information does not arrive on time. The teachers can dynamically edit (compose) the whole structure using a graphical editor (see Figure 5.11 for a snapshot). Many processing nodes have been developed. Table 5.1 summarizes the simplest operations available. Of course designing a new kind of processing node is extremely easy. Computer scientists and musicians program new nodes and arrange them in order to produce interesting musical structures. The children can then control the sound in real time and learn about the structures of the sound while playing.

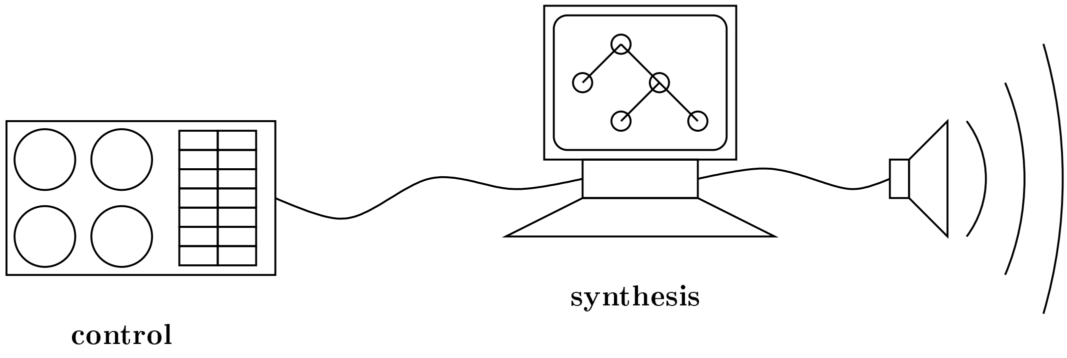


Figure 5.10: The structure of *Dolabip*. The first part (left) is an hardware device sending data to a software tool (right). This tool interprets the control data according to its inner structure – consisting of interconnected processing nodes – and plays the resulting sound. Musical control and sound synthesis are performed in real time.

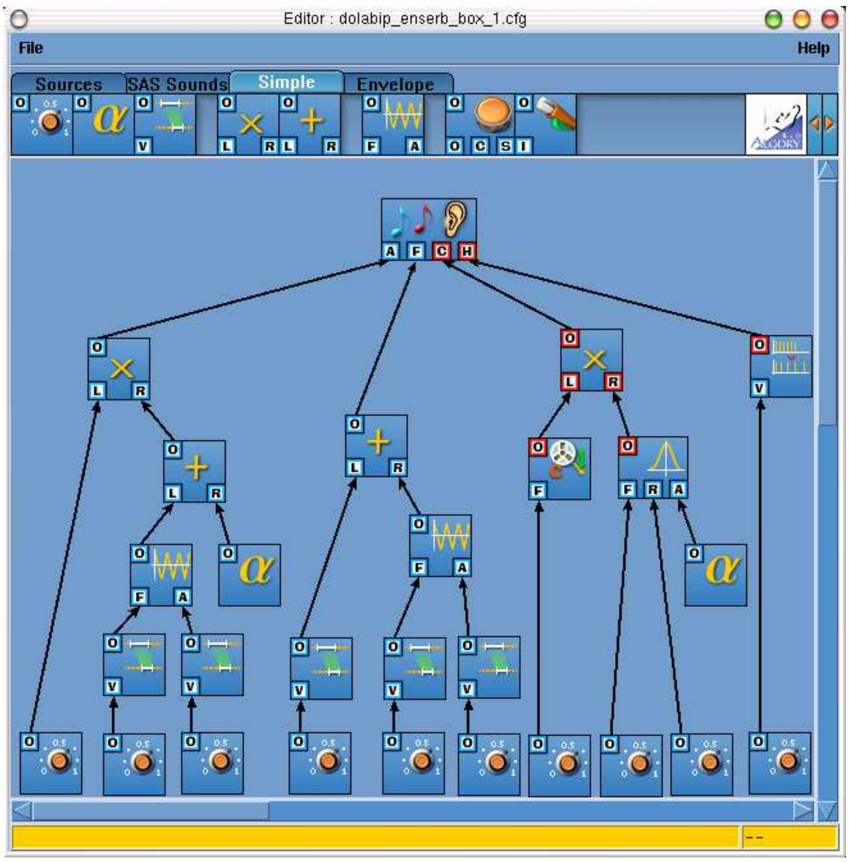


Figure 5.11: Snapshot of the graphical editor of *Dolabip*.

operation	SAS parameter
amplification, tremolo	A
transposition, vibrato	F
filtering, color changing	C
warping	W

Table 5.1: Examples of basic operations proposed in *Dolabip* together with the SAS parameters they modify.

5.4 About Singing Voice

Applications to singing voice are also in progress, since the SAS model is well-suited for the representation of vowels, allowing the control in time of the volume and the pitch as well as the timbre itself. Many synthesis methods for singing voice have been proposed. References can be found in a survey by Georgaki [Geo99a, Geo99b]. These methods often deal with the physical modeling of the vocal tract. Again we propose to focus on the perception of the sound rather than its physical cause.

5.4.1 Advanced Modeling

In the Western musical notation, the continuous flow of sound is quantized in a score, and – a continuum being quantized – information is necessarily lost. The information not carried in the score can be recovered through a set of rules. A performer can use these – sometimes implicit – rules of musical interpretation to reconstitute an acceptable stream of sound. The knowledge that enables human performers to interpret music notation is extremely difficult to represent in a formal way. Synthesizing music from only the information manifested in a traditional score – omitting the information supplied by the performer – results in a wooden rendition which one strains even to call "musical".

Where before music description was mainly considered within the context of musical expression, the computer has now forced us to consider it from a conceptual point of view. Between the composer's imagination and the listener's imagination there is the performer. Clearly, a set of instructions serves to reconstruct the music that the composer originally imagined. We can consider that the role of the musician with respect of the score is to interpret the sediment back into a vivid representation, thus controlling the instrument and producing a stream of parameters for the sound model in charge for the reproduction of sound in the real – physical – world. As a consequence, once a model has been defined for sound, other models have then to be defined also for both the instrument and the performer. Figure 5.12 illustrates this. Such an attempt has already been done, on the top of the SMS sound model, for the saxophone by Ramon López de Mántaras [dM98]. Physical models are modeling instruments but most of them are based on the temporal representation of sound, inheriting from it all its limitations (see Chapter 2).

Once models are defined for sounds, instruments, and performers, then the music notation – often symbolic and discrete by nature – can be translated into a sub-symbolic and continuous flow of control parameters by the performer model, translated in turn into a flow of parameters for the sound model. This advanced modeling approach is very general and we envisage interesting researches on this topic for the near future, especially for the singing voice, where

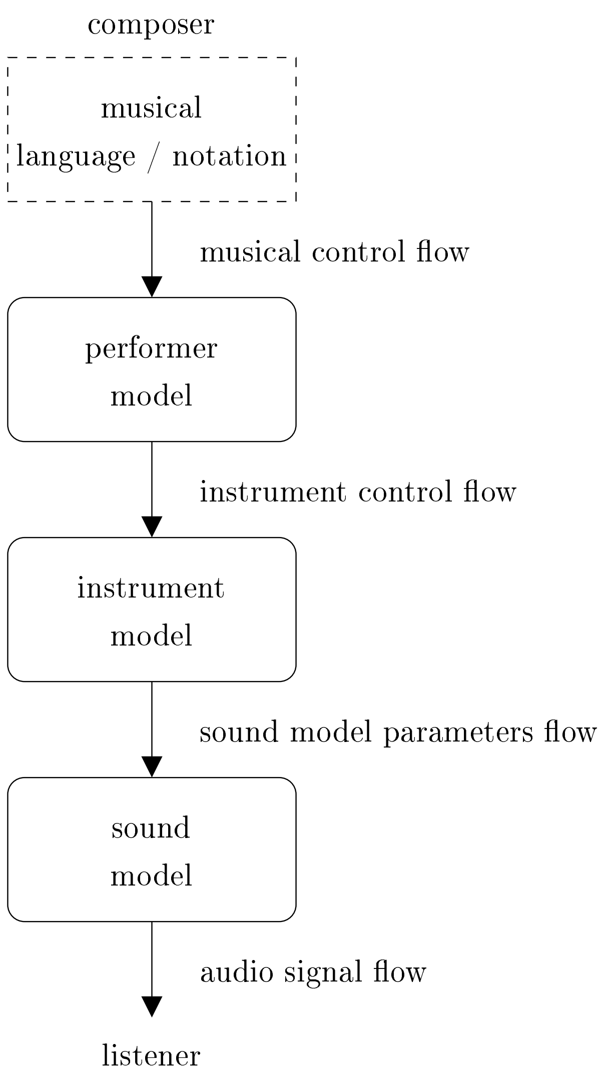


Figure 5.12: Advanced modeling. The composer writes instructions expressed within a musical language – often symbolic and discrete by nature – then translated into a sub-symbolic and continuous flow of control parameters by the performer model, translated in turn into a flow of parameters for the sound model.

the performer and its instrument are, respectively, the singer and the human voice. For example, we aim at designing a program being able to sing simple scores in a realistic way.

5.4.2 Synthesis of Singing Vowels

From a score, we envisage to synthesize a singing voice interpreting this score in a realistic way. We have implemented a software program that extracts the frequency (pitch) and amplitude (volume) parameters contained inside pure musical files, such as MIDI files [MMA96]. The output of this program is a set of (F_i, A_i) pairs faithfully reflecting the evolutions of these parameters for each note of the sequence. The MIDI norm [IMA88, MMA96] has severe dysfunctions [Moo87, Moo88, MM86], especially in terms of expressiveness and timbre control. Several extensions have been proposed for MIDI [IMA91, Rob92, Sch91], as well as a new protocol called ZIPI [McM94, MZW94, Wri94a, Wri94d, Wri94b, Wri94c]. By using the MIDI protocol, our initial evolutions of volume and pitch are crude and lack expressiveness. We have a basic model of the performer (singer), in order to know how a real singer performs the transitions among notes of different pitches and among different vowels, as well as the way tremolo and vibrato may occur as time passes by. After the use of this performer model, we get variations of the frequency and amplitude parameters that are much more realistic. We then use a very simplified model of the human voice. We consider that the color is a function of the vowel and of the fundamental frequency only, thus considering that it does not depend on the amplitude. Although we have already managed to produce satisfactory sounds with this very first attempt of advanced modeling (see above), presenting these results in details will be a little premature. We intend to start a real work on singing voice in the near future. However we briefly present some of our experiments in the remainder of this section.

Vowels. Since a voiced vowel is an harmonic sound, its timbre is totally defined by the color parameter in the SAS model. The difference between two vowels with both the same volume and the same pitch is this color parameter C. Figure 5.13 illustrates this. For a given vowel it appears that its color does not vary much over time if its pitch remains constant. But when the pitch is changing the color usually changes too. Color as a function of the fundamental frequency can be determined from the analysis of recordings. For that purpose we use "sirens", that is vowels with a pitch which is continuously first rising then falling.

Removing/Adding Vibrato/Tremolo. We can remove vibrato and tremolo using the reanalysis of the spectral parameters as indicated in Chapter 3. This removal should be done prior to time-stretching to avoid artifacts. Adding vibrato or tremolo is easier, although we have to take into account the changes in color resulting from the variations of the pitch.

Voiced/Unvoiced Vowels. As mentioned in Chapter 2, it is possible to extend the SAS model to handle noise as well, the amplitude A and color C parameters being still valid. If we generate a noise using these parameters resulting from a voiced vowel (A, F, C, W) in the SAS model, we obtain the same vowel, but this time it is unvoiced.

Transitions among Vowels. We already know, from the analysis, how the transitions should be done for the same vowel but with different pitches. We can then analyze the way these transitions occur among different vowels. More precisely, in order to study the

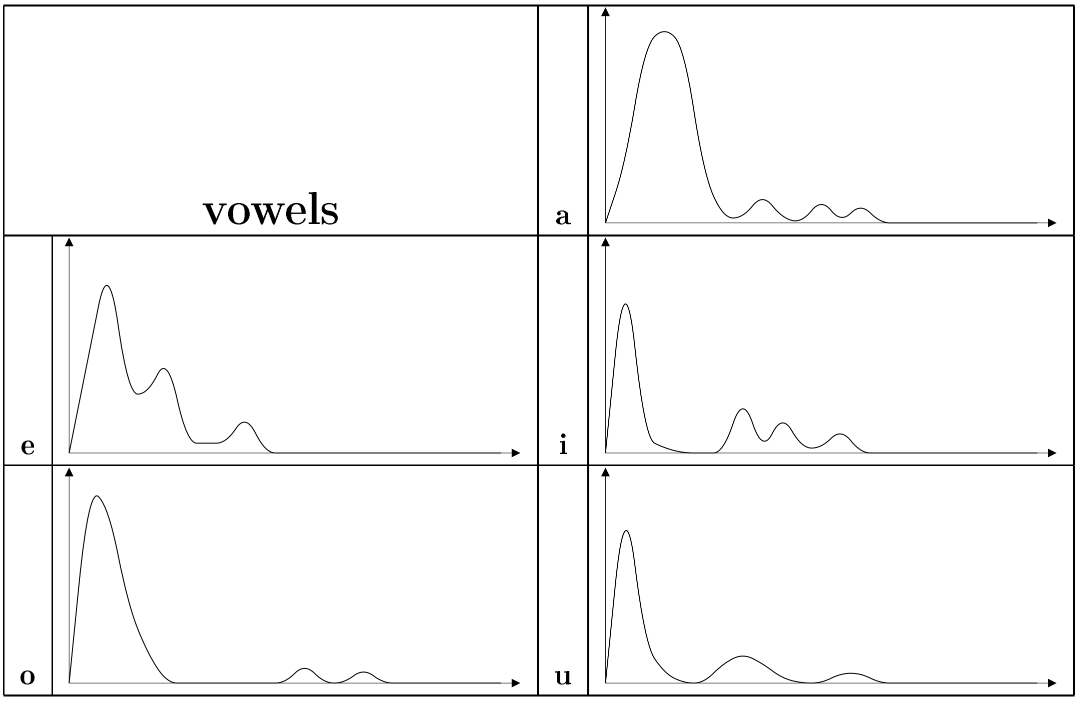


Figure 5.13: Examples of colors for some French vowels (only the evolution with frequency $C(f)$ is represented, since C does not vary much over time in these examples).

transitions among two different vowels, we have analyzed the recordings of a singing voice going from a vowel to another one while keeping the pitch constant. The problem is now to enable simultaneous variations of both pitch and vowel timbre. This cannot be done directly from analysis since the corresponding analysis would require a too large amount of recordings. However we believe that we can deduce these variations from the previous constant-vowel and constant-pitch experiments and by using an appropriate morphing, taking formants into account. Singing voice really provides us with new perspectives of research for the near future.

Conclusions and Future Work

We wanted to be able to reproduce a wide variety of natural sounds – specially instrumental sounds and singing voice – and transform them in a way both musically expressive and computationally efficient. We also needed to be able to play the resulting sounds in real time.

For these purposes, we proposed to focus on spectral sound models based on additive synthesis, and more precisely on sinusoidal models. There were three main problems with these models. First, these models deal with a huge number of parameters involving a large amount of data. The problem is that these parameters are physically valid but only remotely related to musical parameters as perceived by a listener. As a consequence, it is extremely difficult to create or edit realistic sounds within these models. Second, these models need an accurate analysis method in order to represent existing sounds, and the accuracy of the analysis method is extremely important since the perceived quality of the resulting spectral sounds depends mainly on it. The classic short-term Fourier analysis is not enough accurate, and neither are many of its improvements which have been proposed recently. Third, these models require the computation of a large number of sinusoidal oscillators during the synthesis stage. The problem is to find a very fast method for generating the sequence of samples for each oscillator with as few operations as possible. Many implementations are done using hardware devices. However we were looking for an hardware-independent algorithm for real-time synthesis.

In this document, we proposed solutions in order to minimize these problems. First, we have introduced in Chapter 2 the Structured Additive Synthesis (SAS) model. This model imposes constraints on the additive parameters, thus giving birth to structured parameters as close to perception and musical terminology as possible. This model parameters enable to independently modify musical parameters such as pitch, loudness or duration, and constitute as well a solid base for investigating scientific research on the notion of timbre. Since there is a close correspondence between the SAS model parameters and perception, the design and control of musical sound transformations get simplified. Many effects thus become accessible not only to engineers, but also to musicians and composers. Among these effects are time-stretching, cross-syntheses or morphing. Moreover, the SAS model favors the unification between music and sound at a sub-symbolic level [Lem93], thus breaking the arbitrary boundary between sound and music. Second, we have presented in Chapter 3 the most accurate spectral analysis methods. Wavelet-based analyses have been ruled out since they appear to be ill-suited for the musical sound transformations we wanted, mainly because their frequency resolution is so bad for the high frequencies that high harmonics cannot be extracted. Since the classic short-term Fourier analysis was not accurate enough to suit our needs, we have studied the improvements to this analysis that have been proposed recently. Their accuracy was better, but still insufficient in most cases. We proposed to use signal derivatives in order to perform high precision Fourier analysis. Precisely, our method extends the classic short-time

Fourier transform by also considering the signal derivatives, which effectively leads to efficient spectral parameters extraction, and thus allows precise modifications of the inner structures of the sounds. FT^n takes advantage of the first n signal derivatives in order to improve the precision of the Fourier analysis not only in frequency and amplitude but also in time, thus minimizing the problem of the trade-off of time versus frequency in the classic short-time Fourier transform. Third, we have made in Chapter 4 a survey of hardware-independent methods for real-time synthesis. Even if the use of the inverse Fourier transform could be satisfactory, we thought that we could design an even faster synthesis algorithm by using the digital resonator [GS85, SC92]. The problem with this method was to allow the spectral parameters to vary over time while avoiding signal discontinuities as well as numerical imprecision. However we managed to design an algorithm for real-time synthesis based on the digital resonator. It can generate hundreds – if not thousands – of simultaneous sinusoidal oscillators in real time.

As a consequence, we now have a sound model both musically expressive and computationally efficient, together with an accurate analysis method as well as an efficient algorithm for real-time synthesis. Moreover, this model has numerous applications in the fields of creation and education, as shown in Chapter 5. We are even thinking about a model of singing voice based on the SAS model for the near future.

Practical implementations in the field of analysis / transformation / synthesis of spectral sounds are rare. Very few – if any – were available with sources as free software programs. We have developed *InSpect*, *ProSpect*, and *ReSpect* for the purposes of sound analysis, transformation, and synthesis, respectively. All these free software programs are available online [Mar00b]. *ProSpect* is a free software platform that allows the manipulation of sounds in various sound models. It is a research tool also useful for creation. *InSpect* is a software program for sound analysis. We have implemented many analysis methods in the same program, thus providing a very convenient way to make comparisons. We have also compared many synthesis algorithms, and the fastest has been implemented in *ReSpect*, our software tool for real-time sound synthesis. These software programs would certainly deserve to be rewritten as end-user programs and fully documented in order to be usable by non-specialists.

Of course many improvements could still be done. Moreover, some interesting research topics would certainly deserve full studies. Regarding the sound model, it appears that structuring a model in order to facilitate the design of some kinds of sound transformations gives rise to both restrictions on the sounds that can be represented and impossibilities for other kinds of transformations. More precisely, the SAS model can only reproduce monophonic sources with no noise and no transients. Moreover, reverberation, echoing or even the simple mixing of two sounds are impossible to achieve in this model. We believe that adding a hierarchical structuring on the top of our model would ease the manipulation of musical sound. This would make more things possible, like echoing or mixing for example. Moreover, adding noise and possibly transients to the model would allow the modeling of a wider variety of sounds. Regarding analysis accuracy, we are convinced that taking phase distortion into account in our analysis method would improve the accuracy in both frequency and amplitude, thus probably leading to the most accurate analysis method ever proposed so far. Distortion would also provide us with the derivatives of the parameters – frequency and amplitude – of the partials, thus allowing us to design an enhanced partial-tracking strategy. We also believe that the perspective of reanalyzing the spectral parameters themselves is a possible key to very difficult problems such as pitch tracking, lossless compression or even source separation. Regarding synthesis efficiency, we believe that we are already close to the minimal number of operations required per oscillator. We are now thinking about reducing on the fly the number of partials to be synthesized, by taking into account psychoacoustic phenomena such as the different aspects of masking. We also aim at designing an enhanced algorithm especially for modern SIMD (Single Instruction, Multiple Data) architectures.

Let us conclude with a sentence by Risset [Ris88]:

> "Avec l'ordinateur, on peut bâtir un son de structure physique arbitraire :
> mais ce qui importe, c'est son effet sensible."

which means – in English – that although we can build sounds with arbitrary physical structures using a computer, only the way they are perceived really matters. That is why perception is so important. We are convinced that the voice plays a central role [Pay00], and we intend to continue further research on the synthesis of singing voice.

Bibliography

[ACM85] Gérard Assayag, Michèle Castellengo, and Claudy Malherbe. Nouvelles Techniques Instrumentales. Technical Report 38, IRCAM, 1985. In French.

[AD98] Daniel Arfib and Nathalie Delprat. Selective Transformations of Sounds Using Time-Frequency Representations: An Application to the Vibrato Modification. In *104th Convention of the Audio Engineering Society*, Amsterdam, the Netherlands, May 1998. Audio Engineering Society (AES). Preprint 4652 (P5-2).

[AD99] Daniel Arfib and Nathalie Delprat. Alteration of the Vibrato of a Recorded Voice. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 186–189, Beijing, China, October 1999. International Computer Music Association (ICMA).

[AF95] F. Auger and P. Flandrin. Improving the Readibility of Time-Frequency and Time-Scale Representations by the Reassignment Method. *IEEE Transactions on Signal Processing*, 1995.

[AKZ99] Rasmus Althoff, Florian Keiler, and Udo Zölzer. Extracting Sinusoids From Harmonic Signals. pages 97–100, Trondheim, Norway, December 1999. Norwegian University of Science and Technology (NTNU) and COST (European Cooperation in the Field of Scientific and Technical Research).

[All77] Jont B. Allen. Short-Term Spectral Analysis, Synthesis, and Modification by the Discrete Fourier Transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 25(3):235–238, 1977.

[AR77] Jont B. Allen and Lawrence R. Rabiner. A Unified Approach to Short-Time Fourier Analysis and Synthesis. *Proceedings of the IEEE*, 65(11):1558–1564, 1977.

[Arf79] Daniel Arfib. Digital Synthesis of Complex Spectra by Means of Multiplication of Nonlinear Distorted Sine Waves. *Journal of the Audio Engineering Society*, 27(10):757–768, 1979.

[Arf91] Daniel Arfib. *Representation of Musical Signals*, chapter 3: Analysis, Transformation, and Resynthesis of Musical Sounds with the Help of a Time-Frequency Representation, pages 87–118. MIT Press, Cambridge, Massachusetts, 1991.

[Arf98] Daniel Arfib. *Recherches et applications en informatique musicale*, chapter 19 : Des courbes et des sons, pages 277–286. Hermes, Paris, 1998. In French.

[AS90] P. Assmann and Q. Summerfield. Modeling the Perception of Concurrent Vowels: Vowels with Different Fundamental Frequencies. *Journal of the Acoustical Society of America*, 88:680–697, 1990.

[BB95] Karlheinz Brandenburg and Marina Bosi. Overview of MPEG-Audio: Current and Future Standards for Low Bit-Rate Audio Coding. In *99th Convention of the Audio Engineering Society*, New York, October 1995. Audio Engineering Society (AES). Preprint 4130 (29 pages).

[Beu00] Anthony Beurivé. Un logiciel de composition musicale combinant un modèle spectral, des structures hiérarchiques et des contraintes. In *Proceedings of the Journées d'Informatique Musicale (JIM)*, pages 21–30, Bordeaux, May 2000. SCRIME - LaBRI, University of Bordeaux 1. In French.

[BJ95] Robert Bristow-Johnson. A Detailed Analysis of a Time-Domain Formant-Corrected Pitch-Shifting Algorithm. *Journal of the Audio Engineering Society*, 43(5):340–352, 1995.

[BJ99] Stefan Borum and Kristoffer Jensen. Additive Analysis/Synthesis Using Analytically Derived Windows. pages 125–128, Trondheim, Norway, December 1999. Norwegian University of Science and Technology (NTNU) and COST (European Cooperation in the Field of Scientific and Technical Research).

[BL94] Patrice Bourcet and Pierre Liénard. *Le livre des techniques du son*, volume 1, chapter Acoustique Fondamentale, pages 13–43. Éditions Fréquences - Diffusion Eyrolles, Paris, 2nd edition, 1994. In French.

[Bla97] Jens Blauert. *Spatial Hearing*. MIT Press, Cambridge, Massachusetts, revised edition, 1997. Translation by J. S. Allen.

[BP93] J. C. Brown and M. S. Puckette. A High Resolution Fundamental Frequency Determination Based on Phase Changes of the Fourier Transform. *Journal of the Acoustical Society of America*, 94(2):662–667, 1993.

[BPS97] Gianpaolo Borin, Giovanni De Poli, and Augusto Starti. *Musical Signal Processing*, chapter 1: Musical Signal Synthesis, pages 1–30. Studies on New Music Research. Swets & Zeitlinger, Lisse, the Netherlands, 1997.

[BR92] B. Boashash and B. Ristich. Polynomial Wigner-Ville Distributions and Time-Varying Higher Order Spectra. In *Proceedings of the IEEE-SP International Symposium on Time-Frequency and Time-Scale Analysis*, Victoria, Canada, 1992.

[Bro92] J. C. Brown. Musical Fundamental Frequency Tracking Using a Pattern Recognition Method. *Journal of the Acoustical Society of America*, 92(3):1394–1402, 1992.

[BS93] M. Balaban and C. Samoun. Hierarchy, Time and Inheritance in Music Modeling. *Languages of Design*, 1(3), 1993.

[BT58] R. B. Blackman and J. W. Tukey. *The Measurement of Power Spectra*. Dover Publications, New York, 1958.

[BZ92] S. Barbarossa and A. Zanalda. A Combined Wigner-Ville and Hough Transform for Cross-terms Suppression and Optimal Detection and Parameter Estimation. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume V, 1992.

[Car95] Delphine Caron. Modification du son par modification d'image temps-fréquence. DEA Thesis, ATIAM, IRCAM, Paris, 1995. In French.

[Cas82] Michèle Castellengo. Sons Multiphoniques aux Instruments à Vent. Technical Report 34, IRCAM, 1982. In French.

[Cas94] Michèle Castellengo. *Le livre des techniques du son*, volume 1, chapter Les sources acoustiques, pages 45–70. Éditions Fréquences - Diffusion Eyrolles, Paris, 2nd edition, 1994. In French.

[CC98] Nicolas Castagne and Claude Cadoz. Synthèse de sons par modèles physiques dans l'environnement CORDIS-ANIMA. In *Proceedings of the Journées d'Informatique Musicale (JIM)*, page G3, 1998. In French.

[Cha80] G. R. Charbonneau. Timbre and Perceptual Effects of Three Types of Data Reduction. *Computer Music Journal*, 5(2):10–19, 1980.

[Cho73] John M. Chowning. The Synthesis of Complex Audio Spectra by Means of Frequency Modulation. *Journal of the Audio Engineering Society*, 21(7):526–534, 1973.

[Cho97] Andrew Choi. Real-Time Fundamental Frequency Estimation by Least-Square Fitting. *IEEE Transactions on Speech and Audio Processing*, 5(2):201–205, 1997.

[CMD+76] Richard C. Cabot, Michael H. Mino, Douglas A. Dorans, Ira S. Tackel, and Henry E. Breed. Detection of Phase Shifts in Harmonically Related Tones. *Journal of the Audio Engineering Society*, 24(7):568–571, 1976.

[CN96] David Cooper and Kia C. Ng. A Monophonic Pitch-Tracking Algorithm Based on Waveform Periodicity Determinations Using Landmark Points. *Computer Music Journal*, 20(3):70–78, 1996.

[CNM00] CNMAT. SDIF: Sound Description Interchange Format. Online. URL: `http://cnmat.CNMAT.Berkeley.EDU/SDIF/`, 2000.

[CT65] J. W. Cooley and J. W. Tukey. An Algorithm for the Machine Computation of Complex Fourier Series. *Mathematics of Computation*, 19:297–301, 1965.

[Dan93] Roger B. Dannenberg. The Implementation of Nyquist, a Sound Synthesis Language. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 168–171, San Francisco, 1993. International Computer Music Association (ICMA).

[Déc00] François Déchelle. jMax : un environnement pour la réalisation d'applications musicales temps réel sur Linux. In *Proceedings of the Journées d'Informatique Musicale (JIM)*, pages 14–20, Bordeaux, May 2000. SCRIME - LaBRI, University of Bordeaux 1. In French.

[DCM98] Myriam Desainte-Catherine and Sylvain Marchand. High Precision Fourier Analysis of Sounds Using Signal Derivatives. Technical Report 120498, LaBRI, University of Bordeaux 1, 1998. Extended version published in the Journal of the Audio Engineering Society.

[DCM99a] Myriam Desainte-Catherine and Sylvain Marchand. Structured Additive Synthesis: Towards a Model of Sound Timbre and Electroacoustic Music Forms. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 260–263, Beijing, China, October 1999. International Computer Music Association (ICMA).

[DCM99b] Myriam Desainte-Catherine and Sylvain Marchand. Vers un modèle pour unifier musique et son dans une composition multiéchelle. In *Proceedings of the Journées d'Informatique Musicale (JIM)*, pages 59–68, Paris, May 1999. CEMAMu. In French.

[DCM00] Myriam Desainte-Catherine and Sylvain Marchand. High Precision Fourier Analysis of Sounds Using Signal Derivatives. *Journal of the Audio Engineering Society*, 48(7/8):654–667, July/August 2000.

[DCMS99] François Déchelle, Maurizio De Cecco, Enzo Maggi, and Norbert Schnell. jMax Recent Developments. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 445–448, Beijing, China, October 1999. International Computer Music Association (ICMA).

[DCN00] Myriam Desainte-Catherine and Edgard Nicouleau. Segmentation et formalisation d'une oeuvre électroacoustique. To appear in Musurgia, 2000.

[DDH97] Roger B. Dannenberg, Peter Desain, and Henkjan Honing. *Musical Signal Processing*, chapter 8: Programming Language Design for Music, pages 271–315. Studies on New Music Research. Swets & Zeitlinger, Lisse, the Netherlands, 1997.

[Del86] François Delalande. En l'absence de partition. *Analyse Musicale*, pages 54–58, 1986. In French.

[Dem00] Laurent Demany. Les traces d'un son. In *Proceedings of the Journées d'Informatique Musicale (JIM)*, pages 138–144, Bordeaux, May 2000. SCRIME - LaBRI, University of Bordeaux 1. In French.

[DGR93] Philippe Depalle, Guillermo Garcia, and Xavier Rodet. Analysis of Sound for Additive Synthesis: Tracking of Partials Using Hidden Markov Models. In *Proceedings of the International Computer Music Conference (ICMC)*, San Francisco, 1993. International Computer Music Association (ICMA).

[dig] digidesign. *SoftSynth User's Manual*.

[dM98] Ramon López de Mántaras. It Don't Mean A Thing (If It Ain't Got That Swing). In Henri Prade, editor, *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI)*, pages 694–696. John Wiley & Sons, Ltd, 1998.

[Dol86] Mark B. Dolson. The Phase Vocoder: A Tutorial. *Computer Music Journal*, 10(4):14–27, 1986.

[Don99] Matt Donadio. How to Interpolate the Peak Location of a DFT or FFT if the Frequency of Interest is Between Bins. Online. URL: `http://www.dspguru.org/howto/tech/peakfft.htm`, 1999.

[DR93] B. Doval and X. Rodet. Fundamental Frequency Estimation and Tracking Using Maximum Likelihood Harmonic Matching and HMMs. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 221–224, New York, 1993. IEEE.

[DSR87] R. B. Dannenberg, M.-H. Serra, and D. Rubine. Comprehensive Study of Analysis and Synthesis of Tones by Spectral Interpolation. *Journal of the Acoustical Society of America*, 82, supplement 1:S69, 1987.

[Dun91] Andrew Duncan. Combinatorial Music Theory. *Journal of the Audio Engineering Society*, 39(6):21–25, 1991.

[EC98] Gianpaolo Evangelista and Sergio Cavaliere. Dispersive and Pitch-Synchronous Processing of Sounds. In *Proceedings of the Digital Audio Effects (DAFx) Workshop*, pages 232–236, Barcelona, Spain, November 1998. Audiovisual Institute, Pompeu Fabra University and COST (European Cooperation in the Field of Scientific and Technical Research).

[Eva91] Gianpaolo Evangelista. *Representation of Musical Signals*, chapter 4: Wavelet Transform that We Can Play, pages 119–136. MIT Press, Cambridge, Massachusetts, 1991.

[FCS91] Hans G. Feichtinger, C. Cenker, and H. Steier. Fast Iterative and Non-Iterative Reconstruction Methods in Irregular Sampling. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 1773–1776, Toronto, Canada, 1991.

[Fed98] Riccardo Di Federico. Waveform Preserving Time Stretching and Pitch Shifting for Sinusoidal Models of Sound. In *Proceedings of the Digital Audio Effects (DAFx) Workshop*, pages 44–48, Barcelona, Spain, November 1998. Audiovisual Institute, Pompeu Fabra University and COST (European Cooperation in the Field of Scientific and Technical Research).

[FGS95] Hans G. Feichtinger, Karlheinz Gröchenig, and Thomas Strohmer. Efficient Numerical Methods in Non-Uniform Sampling Theory. *Numerische Mathematik*, 69:423–440, 1995.

[FH96] Kelly Fitz and Lippold Haken. Sinusoidal Modeling and Manipulation Using Lemur. *Computer Music Journal*, 20(4):44–59, 1996.

[FJ98] Matteo Frigo and Steven G. Johnson. *FFTW User's Manual*. MIT, Online. URL: `http://www.fftw.org`, 1998.

[FRD92] Adrian Freed, Xavier Rodet, and Philippe Depalle. Synthesis and Control of Hundreds of Sinusoidal Partials on a Desktop Computer without Custom Hardware. In *Proceedings of the ICSPAT'92 Conference*, 1992.

[FRD93] Adrian Freed, Xavier Rodet, and Philippe Depalle. Performance, Synthesis and Control of Additive Synthesis on a Desktop Computer Using FFT^{-1}. In *Proceedings of the International Computer Music Conference (ICMC)*, Tokyo, Japan, 1993. International Computer Music Association (ICMA).

[FS92] Hans G. Feichtinger and Thomas Strohmer. IRSATOL – Irregular Sampling of Band-Limited Signals TOOLBOX. In K. Dette, D. Haupt, and C. Polze, editors, *Computers for Teaching Conference*, pages 277–284, Berlin, Germany, 1992.

[FS93] Hans G. Feichtinger and Thomas Strohmer. Fast Iterative Reconstruction of Band-limited Images from Non-Uniform Sampling Values. In D. Chetverikov and W. G. Kropatsch, editors, *Computer Analysis of Images and Patterns, CAIP Budapest 93 Conference*, pages 82–91, Budapest, Hungary, 1993.

[FSF91] Free Software Fundation FSF. GNU General Public License. Online. URL: `http://www.fsf.org/copyleft/gpl.html`, 1991.

[FvDFH95a] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics – Principles and Practice*, chapter 11: Representing Curves and Surfaces, pages 471–531. System Programming Series. Addison-Wesley, second edition, 1995.

[FvDFH95b] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics – Principles and Practice*, chapter 15: Visible-Surface Determination, pages 649–720. System Programming Series. Addison-Wesley, second (in c) edition, 1995.

[Gab46] Denis Gabor. Theory of Communication. *Journal of the Institute for Electrical Engineers*, 93:429–457, 1946. Part III.

[Gal95] Erick Gallesio. *Extending the STk Interpreter*. Laboratoire I3S, Université de Nice, Sophia Antipolis, Online. URL: `http://kaolin.unice.fr/STk`, July 1995.

[Gal99] Erick Gallesio. *STk Reference Manual*. Laboratoire I3S, Université de Nice, Sophia Antipolis, Online. URL: `http://kaolin.unice.fr/STk`, 4th edition, September 1999.

[Gar92] Guillermo Garcia. Analyse des signaux sonores en termes de partiels et de bruit – Extraction automatique des trajets fréquentiels par des modèles de Markov cachés. DEA Thesis, Université Paris-Sud, Orsay, 1992. In French.

[Geo99a] Anastasia Georgaki. A la recherche de la voix protée : penser la voix de synthèse aujourd'hui. In *Proceedings of the Journées d'Informatique Musicale (JIM)*, pages 97–108, Paris, May 1999. CEMAMu. In French.

[Geo99b] Anastasia Georgaki. Proteïc Voices in the Computer Music Repertory (1972–1997. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 553–556, Beijing, China, October 1999. International Computer Music Association (ICMA).

[Gla95] Andrew S. Glassner. *Principles of Digital Image Synthesis*, volume 1 of *Computer Graphics and Geometric Modeling*, chapter I: The Human Visual System and Color, pages 1–114. Morgan Kaufmann Publishers, Inc, San Francisco, 1995.

[GP99] Guillermo Garcia and Juan Pampin. Data Compression of Sinusoidal Modeling Parameters Based on Psychoacoustic Masking. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 40–43, Beijing, China, October 1999. International Computer Music Association (ICMA).

[Gre75] John M. Grey. *An Exploration of Musical Timbre*. PhD thesis, Department of Music, Stanford University, 1975.

[GS85] J. W. Gordon and J. O. Smith. A Sine Generation Algorithm for VLSI Applications. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 165–168, Vancouver, Canada, 1985.

[GS90] E. Bryan George and Mark J. T. Smith. Analysis-by-Synthesis/Overlap-Add Sinusoidal Modeling Applied to the Analysis and Synthesis of Musical Tones. *Journal of the Audio Engineering Society*, 40(6):497–516, 1990.

[Han96] Peter Hanappe. Intégration des représentations temps/fréquence et des représentations musicales symboliques. In *Proceedings of the Journées d'Informatique Musicale (JIM)*, pages 348–355, Toulon, 1996. In French.

[Han00] Pierre Hanna. Modélisation de bruits. DEA Thesis, LaBRI, Bordeaux, 2000.

[Har78] Fredric J. Harris. On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform. In *Proceedings of the IEEE*, volume 66, pages 51–83, 1978.

[HBEG95] Jürgen Herre, Karlheinz Brandenburg, Ernst Eberlein, and Bernhard Grill. Second Generation ISO/MPEG-Audio Layer III Coding. In *98th Convention of the Audio Engineering Society*, Paris, February 1995. Preprint 3939 (11 pages).

[HF99] Todd Hodes and Adrian Freed. Second-order Recursive Oscillators for Musical Additive Synthesis Applications on SIMD and VLIW Processors. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 74–77, Beijing, China, October 1999. International Computer Music Association (ICMA).

[HI59] L. A. Hiller and L. M. Isaacson. *Experimental Music*. McGraw-Hill, New York, 1959.

[IMA88] IMA. *MIDI 1.0 Detailed Specification*. International MIDI Association (IMA), Los Angeles, 1988.

[IMA91] IMA. *General MIDI System*. International MIDI Association (IMA), Los Angeles, 1991.

[IRC96] IRCAM, Paris. *AudioSculpt User's Manual*, second edition, April 1996.

[IRC00] IRCAM. SDIF – Sound Description Interchange Format. Online. URL: `http://www.ircam.fr/equipes/analyse-synthese/sdif/`, 2000.

[Jac00] Eric Jacobsen. Frequency Estimation Page. Online. URL: `http://www.primenet.com/~ericj/fe.htm`, 2000.

[Jeh97] Tristan Jehan. *Musical Signal Parameter Estimation*. PhD thesis, CNMAT (Berkeley, USA) and IFSIC (Rennes, France), 1997.

[JJS93] Nikil Jayant, James Johnston, and Robert Safranek. Signal Compression Based on Models of Human Perception. In *Proceedings of the IEEE*, volume 81, pages 1385–1421, 1993.

[Kit94] Mpaya Kitantou. *Le livre des techniques du son*, volume 1, chapter La perception auditive, pages 155–181. Éditions Fréquences - Diffusion Eyrolles, Paris, 2nd edition, 1994. In French.

[Kle89] Piotr Kleczkowski. Group Additive Synthesis. *Computer Music Journal*, 13(1):12–20, 1989.

[Koo99] Peter J. Kootsookos. Some Frequency Estimation Algorithms. Online. URL: `http://www.clubi.ie/PeterK/freqalgs.html`, 1999.

[KS80] J. F. Kaiser and R. W. Schafer. On the Use of the I_0-Sinh Window for Spectrum Analysis. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(1):105–107, 1980.

[Kys00] Jaroslav Kysela. Advanced Linux Sound Architecture (ALSA) – The Future for the Linux Sound?! In *Proceedings of the Journées d'Informatique Musicale (JIM)*, pages 1–6, Bordeaux, May 2000. SCRIME - LaBRI, University of Bordeaux 1.

[KZ00] Florian Keller and Udo Zölzer. Extracting Sinusoids from Harmonic Signals. Submitted to Applied Signal Processing, The International Journal of Analog and Digital Signal Processing, 2000.

[LA85] Gareth Loy and Curtis Abbott. Programming Languages for Computer Music Synthesis, Performance, and Composition. *ACM Computing Surveys*, 17(2):235–265, 1985.

[Lan90] John E. Lane. Pitch Detection Using a Tunable IIR Filter. *Computer Music Journal*, 14(3):46–59, 1990.

[Lem93] Marc Leman. *Music Processing*, chapter Symbolic and Subsymbolic Description of Music, pages 119–164. Computer Music and Digital Audio Series. Oxford University Press, Oxford, United Kingdom, 1993.

[Lev98] Scott N. Levine. *Audio Representations for Data Compression and Compressed Domain Processing*. PhD thesis, Department of Electrical Engineering, Stanford University, December 1998.

[LLP99] Fernando Lopez-Lezcano and Juan Pampin. Common Lisp Music Update Report. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 399–402, Beijing, China, October 1999. International Computer Music Association (ICMA).

[Mar98] Sylvain Marchand. Improving Spectral Analysis Precision with an Enhanced Phase Vocoder Using Signal Derivatives. In *Proceedings of the Digital Audio Effects (DAFx) Workshop*, pages 114–118, Barcelona, Spain, November 1998. Audiovisual Institute, Pompeu Fabra University and COST (European Cooperation in the Field of Scientific and Technical Research).

[Mar99] Sylvain Marchand. Musical Sound Effects in the SAS Model. In *Proceedings of the Digital Audio Effects (DAFx) Workshop*, pages 139–142, Trondheim, Norway, December 1999. Norwegian University of Science and Technology (NTNU) and COST (European Cooperation in the Field of Scientific and Technical Research).

[Mar00a] Sylvain Marchand. Compression of Sinusoidal Modeling Parameters. In *Proceedings of the Digital Audio Effects (DAFx) Workshop*, pages 273–276, Verona, Italy, December 2000. Università degli Studi di Verona and COST (European Cooperation in the Field of Scientific and Technical Research).

[Mar00b] Sylvain Marchand. InSpect+ProSpect+ReSpect Software Packages. Online. URL: `http://www.scrime.u-bordeaux.fr/ProSpect`, 2000.

[Mar00c] Sylvain Marchand. InSpect Software Package. Online. URL: `http://www.scrime.u-bordeaux.fr/InSpect`, 2000.

[Mar00d] Sylvain Marchand. Musical Audio Effects in the SAS Model. To appear in Applied Signal Processing, the International Journal of Analog and Digital Signal Processing, 2000.

[Mar00e] Sylvain Marchand. ProSpect : une plate-forme logicielle pour l'exploration spectrale des sons et de la musique. In *Proceedings of the Journées d'Informatique Musicale (JIM)*, pages 31–40, Bordeaux, May 2000. SCRIME - LaBRI, University of Bordeaux 1. In French.

[MB95] Paul Masri and Andrew Bateman. Identification of Nonstationary Audio Signals Using the FFT, with Application to Analysis-based Synthesis of Sound. In *Proceedings IEE Colloquium on Audio Engineering*, pages 11.1–6, London, United Kingdom, 1995. The Institution of Electrical Engineers (IEE).

[MB00] Sylvain Marchand and Anthony Beurivé. Music Composition with Spectral Sounds. Libre Software Meeting (Rencontres Mondiales du Logiciel Libre), Bordeaux, ABUL, URL: `http://www.abul.org`, July 2000.

[MBM99] Stephen McAdams, James W. Beauchamp, and Suzanna Meneguzzi. Discrimination of Musical Instrument Sounds Resynthesized with Simplified Spectrotemporal Parameters. *Journal of the Acoustical Society of America*, 105(2):882–897, 1999.

[MC98] Paul Masri and Nishan Canagarajah. Extracting more detail from the spectrum with Phase Distortion Analysis. In *Proceedings of the Digital Audio Effects (DAFx) Workshop*, pages 119–122, Barcelona, Spain, November 1998. Audiovisual Institute, Pompeu Fabra University and COST (European Cooperation in the Field of Scientific and Technical Research).

[McA84] Stephen McAdams. *Spectral Fusion, Spectral Parsing and the Formation of Auditory Images*. PhD thesis, CCRMA, Department of Music, Stanford University, 1984.

[McM94] Keith McMillen. ZIPI: Origins and Motivations. *Computer Music Journal*, 18(4), 1994.

[MM86] C. Muir and K. McMillen. What's Missing in MIDI? *Guitar Player*, June 1986.

[MMA96] MMA. The Complete MIDI 1.0 Detailed Specification - Standard MIDI Files 1.1. Technical report, MIDI Manufacturers Association, URL: `http://www.midi.org`, 1996.

[Moo77] James A. Moorer. Signal Processing Aspects of Computer Music – A Survey. *Computer Music Journal*, 1(1):4–37, 1977.

[Moo78a] F. Richard Moore. An Introduction to the Mathematics of Digital Signal Processing. *Computer Music Journal*, 2(1):38–47, 1978. Part 1.

[Moo78b] F. Richard Moore. An Introduction to the Mathematics of Digital Signal Processing. *Computer Music Journal*, 2(2):38–60, 1978. Part 2.

[Moo78c] James A. Moorer. The Use of the Phase Vocoder in Computer Music Applications. *Journal of the Audio Engineering Society*, 26(1/2):42–45, 1978.

[Moo85a] F. Richard Moore. *Digital Audio Signal Processing*, chapter An Introduction to the Mathematics of Digital Signal Processing, pages 1–67. The Computer Music and Digital Audio Series. A-R Editions, Inc, Madison, Wisconsin, 1985.

[Moo85b] James A. Moorer. *Digital Audio Signal Processing*, chapter Signal Processing Aspects of Computer Music: A Survey, pages 149–220. The Computer Music and Digital Audio Series. A-R Editions, Inc, Madison, Wisconsin, 1985.

[Moo87] F. Richard Moore. The Dysfunctions of MIDI. In *Proceedings of the International Computer Music Conference (ICMC)*, San Francisco, 1987. Computer Music Association.

[Moo88] F. Richard Moore. The Dysfunctions of MIDI. *Computer Music Journal*, 12(1):19–28, 1988.

[Moo90] F. Richard Moore. *Elements of Computer Music*. Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

[MP80a] Max V. Mathews and John R. Pierce. Harmony and Nonharmonic Partials. Technical Report 28, IRCAM, 1980.

[MP80b] Max V. Mathews and John R. Pierce. Harmony and Nonharmonic Partials. *Journal of the Acoustical Society of America*, 68:1252–1257, 1980.

[MQ86] Robert J. McAulay and Thomas F. Quatieri. Speech Analysis/Synthesis Based on a Sinusoidal Representation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34(4):744–754, 1986.

[MS99] Sylvain Marchand and Robert Strandh. InSpect and ReSpect: Spectral Modeling, Analysis and Real-Time Synthesis Software Tools for Researchers and Composers. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 341–344, Beijing, China, October 1999. International Computer Music Association (ICMA).

[MZW94] Keith McMillen, David L. Zessel, and Matthew Wright. The ZIPI Music Parameter Description Language. *Computer Music Journal*, 18(4):52–73, 1994.

[Nyq28] H. Nyquist. Certain Topics in Telegraph Transmission Theory. *AIEE Transactions*, pages 617–644, 1928.

[OMQ92] R. S. Orr, J. M. Morris, and S. E. Qian. Use of the Gabor Representation for Wigner Distribution Crossterm Suppression. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume V, 1992.

[Orf96] Sophocles J. Orfanidis. *Introduction to Signal Processing*. Signal Processing Series. Prentice-Hall, Englewood Cliffs, New Jersey, 1996.

[OS89] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*. Signal Processing Series. Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[Pam99] Juan Pampin. ATS: A Lisp Environment for Spectral Modeling. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 44–47, Beijing, China, October 1999. International Computer Music Association (ICMA).

[Pan93] Davis Yen Pan. Digital Audio Compression. *Digital Technical Journal*, 5(2):1–13, 1993.

[Pan95] Davis Pan. A Tutorial on MPEG/Audio Compression. *IEEE Multimedia*, 2(2):60–74, 1995.

[Pay00] Blas Payri. Timbre local, timbre global et cohérence du timbre : l'éclairage de la perception de la voix. In *Proceedings of the Journées d'Informatique Musicale (JIM)*, pages 152–161, Bordeaux, May 2000. SCRIME - LaBRI, University of Bordeaux 1.

[PC98] François Pachet and Jean Carrive. *Recherches et applications en informatique musicale*, chapter 1 : Intervalles temporels circulaires et applications à l'analyse harmonique, pages 17–30. Hermes, Paris, 1998. In French.

[PD99] François Pachet and Olivier Delerue. MusicSpace: A Constraint-Based Control System for Music Spatialization. In *Proceedings of the International Computer*

Music Conference (ICMC), pages 272–275, Beijing, China, October 1999. International Computer Music Association (ICMA).

[Pee98] Geoffroy Peeters. Analyse-Synthèse des sons musicaux par la méthode PSOLA. In *Proceedings of the Journées d'Informatique Musicale (JIM)*, Toulon, 1998. In French.

[Pie83] John R. Pierce. *The Science of Musical Sound*. Scientific American Books, Inc, New York, 1983.

[Por76] Michael 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, 1976.

[Por80] Michael R. Portnoff. Time-Frequency Representation of Digital Signals and Systems Based on Short-Time Fourier Transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(8):55–69, 1980.

[PPK99] D. Pressnitzer, R. D. Patterson, and K. Krumbholz. The Lower Limit of Melodic Pitch with Filtered Harmonic Complexes. *Journal of the Acoustical Society of America*, 105:1152, 1999.

[PR98] Geoffroy Peeters and Xavier Rodet. Signal Characterization in terms of Sinusoidal and Non-Sinusoidal Components. In *Proceedings of the Digital Audio Effects (DAFx) Workshop*, pages 123–126, Barcelona, Spain, November 1998. Audiovisual Institute, Pompeu Fabra University and COST (European Cooperation in the Field of Scientific and Technical Research).

[PR99] Geoffroy Peeters and Xavier Rodet. SINOLA: A New Analysis/Synthesis Method using Spectrum Peak Shape Distortion, Phase and Reassigned Spectrum. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 153–156, Beijing, China, October 1999. International Computer Music Association (ICMA).

[Pre00] Daniel Pressnitzer. Modèles psychoacoustiques et perception de hauteur. In *Proceedings of the Journées d'Informatique Musicale (JIM)*, pages 145–151, Bordeaux, May 2000. SCRIME - LaBRI, University of Bordeaux 1. In French.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C (The Art of Scientific Computing)*, chapter 10: Minimization or Maximization of Functions, pages 402–405. Cambridge University Press, USA, 2nd edition, 1992.

[Qui94] Barry G. Quinn. Estimating Frequency by Interpolation Using Fourier Coefficients. *IEEE Transactions on Signal Processing*, 42(5):1264–1268, 1994.

[Rab77] Lawrence R. Rabiner. On the Use of Autocorrelation Analysis for Pitch Detection. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 25(1):24–33, 1977.

[Ram65] J.-P. Rameau. *Traité de l'harmonie réduite à ses principes naturels*. Broude Brothers, New York, facsimile of 1722 paris edition, 1965.

[Rio84] André Riotte. Un modèle informatique pour la transformation continue de sons inharmoniques. In W. Buxton, editor, *Proceedings of the International Computer Music Conference (ICMC)*, San Francisco, 1984. Computer Music Association.

[Ris86a] Jean-Claude Risset. Musical Sound Models for Digital Signals. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 1269–1271, New York, 1986. IEEE.

[Ris86b] Jean-Claude Risset. Timbre et synthèse de sons. *Analyse musicale*, pages 9–19, 1986. In French.

[Ris88] Jean-Claude Risset. Perception, environnement, musiques. *Harmoniques*, 2:10–42, 1988. In French.

[Ris91] Jean-Claude Risset. *Representation of Musical Signals*, chapter 1: Timbre Analysis by Synthesis: Representations, Imitations, and Variants for Musical Composition, pages 7–43. MIT Press, Cambridge, Massachusetts, 1991.

[Riv99] Jean-Michel Rivet. Sept Couronnes pour Goethe (Fragment 4). Sonoris, publisher, 1999. Musical Work.

[RM69] Jean-Claude Risset and Max V. Mathews. Analysis of Musical Instrument Tones. *Physics Today*, 22(2):23–30, 1969.

[Roa96] Curtis Roads. *The Computer Music Tutorial.* MIT Press, Cambridge, Massachusetts, 1996.

[Rob92] A. Roberts. Devices for Increasing the Number of MIDI Channels. *Computer Music Journal*, 16(4):101–104, 1992.

[Rév44] G. Révész. Inleiding tot de Muziekpsychologie. N. V. Noord-Hollandsche Uitgevers Maatschappij, Amsterdam, the Netherlands, 1944. In Dutch.

[SC92] Julius O. Smith and Perry R. Cook. The Second-Order Digital Waveguide Oscillator. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 150–153, San Jose, 1992.

[Sca87] Carla Scaletti. Kyma: an Object-Oriented Language for Music Composition. In James Beauchamp, editor, *Proceedings of the International Computer Music Conference (ICMC)*, pages 49–56, San Francisco, 1987. Computer Music Association.

[Sca89] Carla Scaletti. The Kyma/Platypus Computer Music Workstation. *Computer Music Journal*, 13(2):23–38, 1989.

[Sch91] C. Scholz. A Proposed Extension to the MIDI Specification Concerning Tuning. *Computer Music Journal*, 15(1):49–54, 1991.

[Sch94] Pierre Schaeffer. *Traité des Objets Musicaux.* Oxford University Press, 1994.

[SCR99] SCRIME. Examples of Morphing in the SAS Model (Presented at DAFx'99). Online. URL: `http://www.scrime.u-bordeaux.fr/DAFx99`, 1999.

[SD91] Catherine Semal and Laurent Demany. Dissociation of Pitch from Timbre in Auditory Short-Term Memory. *Journal of the Acoustical Society of America*, 89:2404–2410, 1991.

[SD93] Catherine Semal and Laurent Demany. Further Evidence for an Autonomous Processing of Pitch in Auditory Short-Term Memory. *Journal of the Acoustical Society of America*, 94:1315–1322, 1993.

[Ser89] Xavier Serra. *A System for Sound Analysis/Transformation/Synthesis Based on a Deterministic plus Stochastic Decomposition.* PhD thesis, CCRMA, Department of Music, Stanford University, 1989.

[Ser97a] Marie-Hélène Serra. *Musical Signal Processing*, chapter 2: Introducing the Phase Vocoder, pages 31–90. Studies on New Music Research. Swets & Zeitlinger, Lisse, the Netherlands, 1997.

[Ser97b] Xavier Serra. *Musical Signal Processing*, chapter 3: Musical Sound Modeling with Sinusoids plus Noise, pages 91–122. Studies on New Music Research. Swets & Zeitlinger, Lisse, the Netherlands, 1997.

[SG84] Julius O. Smith and Phil Gossett. A Flexible Sampling-Rate Conversion Method. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 2, pages 19.4.1–19.4.2, San Diego, 1984. IEEE.

[Sha49] C. E. Shannon. Communication in the Presence of Noise. In *Proceedings IRE*, pages 10–12, 1949.

[She82] Roger N. Shepard. *The Psychology of Music*, chapter Structural Representations of Musical Pitch. Academic Press, New York, 1982.

[She83] Roger N. Shepard. Demonstrations of Circular Components of Pitch. *Journal of the Audio Engineering Society*, 31(7):641–649, 1983.

[SJ89] Hajime Sano and B. Keith Jenkins. A Neural Network Model for Pitch Perception. *Computer Music Journal*, 13(3):41–48, 1989.

[SM99] Robert Strandh and Sylvain Marchand. Real-Time Generation of Sound from Parameters of Additive Synthesis. In *Proceedings of the Journées d'Informatique Musicale (JIM)*, pages 83–88, Paris, May 1999. CEMAMu.

[Smi85] Julius O. Smith. *Digital Audio Signal Processing*, chapter An Introduction to Digital Filter Theory, pages 69–135. The Computer Music and Digital Audio Series. A-R Editions, Inc, Madison, Wisconsin, 1985.

[Smi00] Julius O. Smith. Digital Audio Resampling Home Page. Technical report, CCRMA, Online. URL: `http://www-ccrma.stanford.edu/~jos/resample/resample.html`, 2000.

[SR99] Diemo Schwarz and Xavier Rodet. Spectral Envelope Estimation and Representation for Sound Analysis-Synthesis. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 351–354, Beijing, China, October 1999. International Computer Music Association (ICMA).

[SRD] M.-H. Serra, D. Rubine, and R. B. Dannenberg. The Analysis and Resynthesis of Tones via Spectral Interpolation. In C. Lischka and J. Fritsch, editors, *Proceedings of the 14th International Computer Music Conference*, San Francisco. Computer Music Association.

[SS87] Julius O. Smith and Xavier Serra. PARSHL: An Analysis/Synthesis Program for Non-Harmonic Sounds based on a Sinusoidal Representation. In James Beauchamp, editor, *Proceedings of the International Computer Music Conference (ICMC)*, San Francisco, 1987. Computer Music Association.

[SS90] Xavier Serra and Julius O. 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] John Strawn. Approximation and Syntactic Analysis of Amplitude and Frequency Functions for Digital Sound Synthesis. *Computer Music Journal*, 4(3), 1980.

[Str87] John Strawn. Editing Time-Varying Spectra. *Journal of the Audio Engineering Society*, 35(5):337–352, 1987.

[Str93] Thomas Strohmer. *Efficient Methods for Digital Signal and Image Reconstruction from Nonuniform Samples.* PhD thesis, University of Vienna, Austria, 1993.

[Str97] Thomas Strohmer. Computationally Attractive Reconstruction of Band-Limited Images from Irregular Samples. *IEEE Transactions on Image Processing*, 6(4):540–548, 1997.

[Tod96] Todor Todoroff. Instrument de transformation par analyse-synthèse dans Max-FTS. In *Proceedings of the Journées d'Informatique Musicale (JIM)*, pages 103–110, Toulon, 1996. In French.

[Vag94] Horacio Vaggione. Timbre as Syntax: A Spectral Modeling Approach. *Contemporary Music Review*, 8(2):91–104, 1994. In S. Emmerson, editor. Timbre in Electro-Acoustic Music Composition.

[Vag96] Horacio Vaggione. Articulating Microtime. *Computer Music Journal*, 20(2):33–38, 1996.

[Vag98] Horacio Vaggione. Transformations morphologiques : quelques exemples. In *Proceedings of the Journées d'Informatique Musicale (JIM)*, page G1, Toulon, 1998. In French.

[Ver86] Barry L. Vercoe. *CSound: A Manual for the Audio Processing System and Supporting Programs.* MIT Media Laboratory, Cambridge, Massachusetts, 1986.

[Ver92] Barry L. Vercoe. *The CSound Manual Version 3.48. A Manual for the Audio Processing System and Supporting Softwares with Tutorials.* MIT Media Laboratory, Cambridge, Massachusetts, 1992. Edited by Jean Piché, University of Montreal. URL: `ftp://ftp.musique.umontreal.ca/pub`.

[VGS98] B. L. Vercoe, W. G. Gardner, and E. D. Scheirer. Structured Audio: The Creation, Transmission, and Rendering of Parametric Sound Representations. *Proceedings of the IEEE*, 86(5):922–940, 1998.

[vH54] Hermann von Helmholtz. *On the Sensations of Tone as a Physiological Basis for the Theory of Music*. Dover Publications, New York, 1954. Translation of the 1877 edition by Alexander J. Ellis.

[VM98a] Tony S. Verma and Teresa H.Y. Meng. An Analysis/Synthesis Tool for Transient Signals that Allows a Flexible Sines+Transients+Noise Model for Audio. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE, 1998.

[VM98b] Tony S. Verma and Teresa H.Y. Meng. Time Scale Modification Using a Sines+Transients+Noise Signal Model. In *Proceedings of the Digital Audio Effects (DAFx) Workshop*, pages 49–52, Barcelona, Spain, November 1998. Audiovisual Institute, Pompeu Fabra University and COST (European Cooperation in the Field of Scientific and Technical Research).

[WCF+99] Matthew Wright, Amar Chaudhary, Adrian Freed, Sami Khoury, and David Wessel. Audio Applications of the Sound Description Interchange Format Standard. *107th Convention of the Audio Engineering Society*, September 1999.

[Wes78] David L. Wessel. Low Dimensional Control of Musical Timbre. Technical Report 12, IRCAM, 1978.

[Wes79] David L. Wessel. Timbre Space as a Musical Control Structure. *Computer Music Journal*, 3(2):45–52, 1979.

[WL24] R. L. Wegel and C. E. Lane. The Auditory Masking of One Pure Tone by Another and Its Probable Relation to the Dynamics of the Inner Ear. *Physical Review*, 23:266–285, 1924.

[Wri94a] Matthew Wright. A Comparison of MIDI and ZIPI. *Computer Music Journal*, 18(4):86–91, 1994.

[Wri94b] Matthew Wright. A Summary of the ZIPI Network. *Computer Music Journal*, 18(4):74–80, 1994.

[Wri94c] Matthew Wright. Answers to Frequently Asked Questions About ZIPI. *Computer Music Journal*, 18(4):92–96, 1994.

[Wri94d] Matthew Wright. Examples of ZIPI Applications. *Computer Music Journal*, 18(4):81–85, 1994.

[wwwa] ALSA (Advanced Linux Sound Architecture). Online. URL: `http://www.alsa-project.org`.

[wwwb] OSS (Open Sound System) Free. Online. URL: `http://www.linux.org.uk/OSS`.

[ZF81] Eberhard Zwicker and Richard Feldtkeller. *Psychoacoustique – L'oreille, récepteur d'information*. Masson, Paris, 1981. In French.

[ZF90] E. Zwicker and H. Fastl. *Psychoacoustics Facts and Models*. Springer Verlag, 1990.

[ZZ91] Eberhard Zwicker and Ulrich Tilmann Zwicker. Audio Engineering and Psychoacoustics: Matching Signals to the Final Receiver, the Human Auditory System. *Journal of the Audio Engineering Society*, 39(3):115–126, 1991.

(*Total:* 214 references)

Appendix A

Numerical Results from Analysis

This appendix lists the numerical results referenced in Chapter 3. All the short-time analysis methods use a sliding window with a hop size of 64 samples. The sampling rate is 44100 Hz. The following formulae correspond to the synthetic examples used for the tests.

oscillator with linearly-increasing frequency:

$$a(t) = A_0 \sin\left(2\pi\left(F_1 + \frac{(F_2 - F_1)}{D}t\right)t\right)$$

- $D = 5$ s (duration)
- $A_0 = 0.8$ (base amplitude)
- $F_1 = 440$ Hz (from frequency)
- $F_2 = 1660$ Hz (to frequency)

oscillator with vibrato:

$$a(t) = A_0 \sin\left(2\pi\left(F_0 + A_1 \sin(2\pi F_1 t)\right)t\right)$$

- $A_0 = 1$ (base amplitude)
- $F_0 = 2000$ Hz (base frequency)
- A_1: vibrato amplitude (depth)
- F_1: vibrato frequency (rate)

oscillator with tremolo:

$$a(t) = (A_0 + A_1 \sin(2\pi F_1 t))\sin(2\pi F_0 t)$$

- $A_0 = 0.5$ (base amplitude)
- $F_0 = 2000$ Hz (base frequency)
- A_1: tremolo amplitude (depth)
- F_1: tremolo frequency (rate)

window		DFT	Brent	DFT[1]
rectangular	256	4.67 (50.0615)	2.12 (21.6893)	3.01 (36.2808)
	512	2.36 (24.8117)	1.05 (10.5738)	1.52 (17.8342)
	1024	1.18 (12.4679)	0.52 (5.2677)	0.76 (8.8946)
	2048	0.58 (6.2141)	0.27 (2.7187)	0.38 (4.4740)
Hann	256	4.66 (49.9861)	0.19 (1.9592)	0.04 (0.4101)
	512	2.36 (24.7995)	0.10 (0.9851)	0.01 (0.0890)
	1024	1.18 (12.4665)	0.05 (0.4918)	0.00 (0.0632)
	2048	0.58 (6.2138)	0.02 (0.2334)	0.01 (0.1207)
Hamming	256	4.66 (49.9925)	0.19 (1.9623)	0.36 (4.2860)
	512	2.36 (24.8003)	0.10 (0.9839)	0.19 (2.2264)
	1024	1.18 (12.4666)	0.05 (0.4893)	0.10 (1.1356)
	2048	0.58 (6.2139)	0.02 (0.2163)	0.05 (0.5945)
Blackman	256	4.66 (49.9860)	0.09 (0.8684)	0.02 (0.1935)
	512	2.36 (24.7995)	0.04 (0.4339)	0.01 (0.0725)
	1024	1.18 (12.4665)	0.02 (0.2042)	0.00 (0.0548)
	2048	0.58 (6.2138)	0.01 (0.0955)	0.01 (0.0915)
Gauss	256	4.66 (49.9860)	0.01 (0.1067)	0.01 (0.1307)
	512	2.36 (24.7995)	0.00 (0.0035)	0.00 (0.0648)
	1024	1.18 (12.4665)	0.00 (0.0030)	0.00 (0.0427)
	2048	0.58 (6.2138)	0.00 (0.0028)	0.00 (0.0496)

Table A.1: Error on frequency measured by the DFT without peak interpolation, the DFT plus parabolic peak interpolation using the Brent method, and the DFT[1] method. The average error on frequency in percents (as well as the standard deviation) is given for different analysis window types and widths. The analyzed signal is a single sinusoidal oscillator whose frequency is linearly increasing from 440 Hz to 1660 Hz while its amplitude remains constant at 0.8. This sound lasts 5 s and $F_s = 44100$ Hz.

window		DFT	Brent	DFT[1]
rectangular	256	12.82 (0.1355)	10.62 (0.1097)	11.14 (0.1229)
	512	12.66 (0.1342)	10.52 (0.1089)	11.05 (0.1223)
	1024	12.84 (0.1354)	10.68 (0.1100)	11.00 (0.1209)
Hann	256	5.19 (0.0550)	1.30 (0.0138)	0.09 (0.0011)
	512	5.13 (0.0547)	1.29 (0.0137)	0.06 (0.0007)
	1024	5.20 (0.0552)	1.30 (0.0138)	0.09 (0.0010)
Hamming	256	6.33 (0.0670)	1.60 (0.0170)	0.98 (0.0114)
	512	6.25 (0.0665)	1.59 (0.0169)	1.00 (0.0116)
	1024	6.34 (0.0671)	1.58 (0.0168)	1.04 (0.0119)
Blackman	256	4.06 (0.0431)	0.46 (0.0044)	0.05 (0.0005)
	512	4.01 (0.0429)	0.45 (0.0044)	0.05 (0.0005)
	1024	4.07 (0.0433)	0.38 (0.0040)	0.07 (0.0007)
Gauss	256	2.26 (0.0241)	0.00 (0.0001)	0.02 (0.0003)
	512	2.23 (0.0238)	0.00 (0.0000)	0.02 (0.0002)
	1024	2.25 (0.0240)	0.00 (0.0000)	0.03 (0.0003)

Table A.2: Error on amplitude measured by the DFT without peak interpolation, the DFT plus parabolic peak interpolation using the Brent method, and the DFT[1] method. The average error on amplitude in percents (as well as the standard deviation) is given for different analysis window types and widths. The analyzed signal is the same as in Table A.1.

	window width	Triangular	DFT[1]
Frequency	512	0.02 (0.1596)	0.01 (0.1251)
	1024	0.01 (0.0780)	0.00 (0.0496)
Amplitude	512	0.00 (0.0000)	0.03 (0.0004)
	1024	0.00 (0.0000)	0.04 (0.0004)

Table A.3: Error on frequency (top) and amplitude (bottom) measured by the triangular algorithm and the DFT[1] method. The average error in percents (as well as the standard deviation) is given for the triangular-frequency analysis window with different widths, thus favoring the triangular algorithm. The analyzed signal is the same as in Tables A.1 and A.2.

window	noise level (SNR)	Brent	DFT[1]
Hann 512	0% (∞ dB)	0.10 (0.9851)	0.01 (0.0890)
	12.5% (18 dB)	0.10 (1.0119)	0.02 (0.2199)
	25% (12 dB)	0.10 (1.0979)	0.04 (0.4155)
	50% (6 dB)	0.12 (1.3796)	0.07 (0.8062)
	100% (0 dB)	104.68 (1148.1205)	0.14 (1.5677)
	200% (-6 dB)	104.68 (1148.1205)	0.31 (3.2974)
Gauss 512	0% (∞ dB)	0.00 (0.0035)	0.00 (0.0648)
	12.5% (18 dB)	0.03 (0.3922)	0.03 (0.3836)
	25% (12 dB)	0.07 (0.7820)	0.06 (0.7543)
	50% (6 dB)	0.13 (1.6027)	0.13 (1.5451)
	100% (0 dB)	104.71 (1148.1205)	0.28 (2.9523)
	200% (-6 dB)	104.68 (1148.1205)	0.57 (7.0350)

Table A.4: Error on frequency measured by the Brent and the DFT[1] methods. The average error on frequency in percents (as well as the standard deviation) is given for two different analysis windows. The analyzed signal is the same as in Table A.1, but white noise has been added.

window	noise level (SNR)	Brent	DFT[1]
Hann 512	0% (∞ dB)	1.29 (0.0137)	0.06 (0.0007)
	12.5% (18 dB)	1.36 (0.0140)	0.27 (0.0027)
	25% (12 dB)	1.45 (0.0148)	0.51 (0.0053)
	50% (6 dB)	1.82 (0.0174)	1.06 (0.0104)
	100% (0 dB)	33.54 (0.2706)	2.14 (0.0213)
	200% (-6 dB)	166.99 (1.3387)	4.34 (0.0411)
Gauss 512	0% (∞ dB)	0.00 (0.0000)	0.02 (0.0002)
	12.5% (18 dB)	0.35 (0.0035)	0.35 (0.0035)
	25% (12 dB)	0.69 (0.0065)	0.69 (0.0068)
	50% (6 dB)	1.39 (0.0137)	1.39 (0.0137)
	100% (0 dB)	26.66 (0.2181)	2.67 (0.0274)
	200% (-6 dB)	153.49 (1.2308)	5.20 (0.0526)

Table A.5: Error on amplitude measured by the Brent and the DFT[1] methods. The average error on amplitude in percents (as well as the standard deviation) is given for two different analysis windows. The analyzed signal is the same as in Table A.1, but white noise has been added.

	noise level (SNR)	Triangular	DFT[1]
Frequency	0% (∞ dB)	0.02 (0.1596)	0.01 (0.1251)
	12.5% (18 dB)	0.08 (0.8404)	0.05 (0.5247)
	25% (12 dB)	0.15 (1.6040)	0.09 (1.0478)
	50% (6 dB)	0.30 (3.2333)	0.19 (2.0833)
	100% (0 dB)	0.54 (6.1822)	0.40 (4.1836)
	200% (-6 dB)	0.95 (11.4686)	0.75 (9.0680)
Amplitude	0% (∞ dB)	0.00 (0.0000)	0.03 (0.0004)
	12.5% (18 dB)	0.45 (0.0046)	0.37 (0.0036)
	25% (12 dB)	0.89 (0.0089)	0.73 (0.0074)
	50% (6 dB)	1.80 (0.0176)	1.49 (0.0151)
	100% (0 dB)	3.49 (0.0359)	3.01 (0.0298)
	200% (-6 dB)	13.90 (0.0720)	5.93 (0.0602)

Table A.6: Error on frequency (top) and amplitude (bottom) measured by the triangular algorithm and the DFT[1] method. The average error in percents (as well as the standard deviation) is given for the 512-point triangular-frequency analysis window, thus favoring the triangular algorithm. The analyzed signal is the same as in Table A.1, but white noise has been added.

Brent	0	10	100	500	1000
5	0.06 (1.2567)	0.06 (1.1570)	0.05 (0.9948)	0.03 (0.8006)	0.05 (1.1319)
10	0.06 (1.2567)	0.06 (1.1514)	0.05 (1.0003)	0.09 (1.9749)	0.14 (3.0581)
15	0.06 (1.2567)	0.06 (1.1438)	0.05 (1.1928)	0.16 (3.8312)	0.47 (9.2539)
20	0.06 (1.2567)	0.06 (1.1368)	0.07 (1.6800)	0.24 (5.9975)	0.97 (19.6803)
25	0.06 (1.2567)	0.06 (1.1341)	0.11 (2.4322)	0.46 (10.1610)	1.73 (35.2973)

DFT[1]	0	10	100	500	1000
5	0.01 (0.1278)	0.01 (0.1434)	0.01 (0.3612)	0.09 (2.5012)	0.33 (8.5786)
10	0.01 (0.1278)	0.01 (0.1689)	0.04 (0.8669)	0.34 (8.9992)	0.86 (19.2479)
15	0.01 (0.1278)	0.01 (0.2235)	0.07 (1.6689)	0.63 (15.1601)	1.14 (24.2318)
20	0.01 (0.1278)	0.01 (0.3108)	0.12 (2.7400)	0.87 (19.9062)	1.52 (32.3986)
25	0.01 (0.1278)	0.02 (0.4290)	0.18 (4.0701)	1.07 (24.3429)	2.19 (46.7466)

Table A.7: Error on frequency measured by the Brent and the DFT[1] methods. The analyzed signal is a single sinusoidal oscillator whose frequency is 2000 Hz modulated by a vibrato while its amplitude remains constant at 1. This sound lasts 5 s and $F_s = 44100$ Hz. The average error on frequency in percents (as well as the standard deviation) is given for different vibrato rates and depths. The analysis window is the 512-point Hann window. A boundary indicates when the error is greater than 0.1%.

Brent	0	10	100	500	1000
5	0.00 (0.0027)	0.00 (0.0062)	0.00 (0.0611)	0.01 (0.3068)	0.03 (0.5931)
10	0.00 (0.0027)	0.00 (0.0224)	0.01 (0.2281)	0.05 (1.1077)	0.10 (1.9568)
15	0.00 (0.0027)	0.00 (0.0500)	0.02 (0.5041)	0.11 (2.3255)	0.18 (3.7395)
20	0.00 (0.0027)	0.00 (0.0884)	0.04 (0.8860)	0.17 (3.8389)	0.26 (5.8628)
25	0.00 (0.0027)	0.01 (0.1375)	0.06 (1.3700)	0.23 (5.5375)	0.39 (8.5628)

DFT[1]	0	10	100	500	1000
5	0.01 (0.1278)	0.01 (0.1393)	0.01 (0.2228)	0.03 (0.8780)	0.09 (2.2849)
10	0.01 (0.1278)	0.01 (0.1478)	0.02 (0.4120)	0.10 (2.4800)	0.33 (8.7983)
15	0.01 (0.1278)	0.01 (0.1658)	0.03 (0.7333)	0.20 (4.6218)	0.56 (12.8834)
20	0.01 (0.1278)	0.01 (0.1964)	0.05 (1.1742)	0.36 (9.5308)	0.77 (16.6990)
25	0.01 (0.1278)	0.01 (0.2414)	0.08 (1.7355)	0.49 (11.9566)	0.94 (19.4474)

Table A.8: Same comparison as in Table A.7, but this time with a truncated Gaussian window.

Brent	0	10	100	500	1000
5	0.78 (0.0078)	0.88 (0.0105)	1.10 (0.0148)	1.61 (0.0192)	6.26 (0.0757)
10	0.78 (0.0078)	0.88 (0.0105)	0.92 (0.0123)	6.26 (0.0754)	18.33 (0.2150)
15	0.78 (0.0078)	0.88 (0.0105)	0.76 (0.0099)	12.35 (0.1453)	27.96 (0.3175)
20	0.78 (0.0078)	0.87 (0.0105)	0.87 (0.0111)	18.24 (0.2117)	34.39 (0.3829)
25	0.78 (0.0078)	0.87 (0.0105)	1.38 (0.0168)	23.22 (0.2661)	38.90 (0.4276)

DFT[1]	0	10	100	500	1000
5	0.09 (0.0009)	0.08 (0.0009)	0.18 (0.0026)	2.22 (0.0288)	7.38 (0.0897)
10	0.09 (0.0009)	0.08 (0.0009)	0.44 (0.0059)	7.34 (0.0893)	19.50 (0.2256)
15	0.09 (0.0009)	0.08 (0.0010)	0.86 (0.0111)	13.45 (0.1585)	28.79 (0.3232)
20	0.09 (0.0009)	0.09 (0.0011)	1.40 (0.0177)	19.24 (0.2219)	35.16 (0.3867)
25	0.09 (0.0009)	0.10 (0.0012)	2.07 (0.0258)	24.13 (0.2735)	39.56 (0.4305)

Table A.9: Error on amplitude measured by the Brent and the DFT[1] methods. The analyzed signal and analysis window are the same as in Table A.7. The average error on amplitude in percents (as well as the standard deviation) is given for different vibrato rates and depths. A boundary indicates when the error is greater than 0.1% (which is always the case here for the Brent method).

Brent	0	10	100	500	1000
5	0.00 (0.0000)	0.00 (0.0000)	0.02 (0.0002)	0.42 (0.0051)	1.60 (0.0195)
10	0.00 (0.0000)	0.00 (0.0000)	0.07 (0.0008)	1.60 (0.0194)	5.49 (0.0657)
15	0.00 (0.0000)	0.00 (0.0000)	0.15 (0.0018)	3.34 (0.0403)	10.14 (0.1193)
20	0.00 (0.0000)	0.00 (0.0000)	0.27 (0.0032)	5.43 (0.0649)	14.68 (0.1700)
25	0.00 (0.0000)	0.01 (0.0001)	0.41 (0.0049)	7.68 (0.0909)	18.79 (0.2146)

DFT^1	0	10	100	500	1000
5	0.04 (0.0004)	0.03 (0.0004)	0.06 (0.0007)	0.50 (0.0060)	1.81 (0.0224)
10	0.04 (0.0004)	0.03 (0.0004)	0.11 (0.0014)	1.78 (0.0219)	5.88 (0.0704)
15	0.04 (0.0004)	0.03 (0.0004)	0.20 (0.0025)	3.65 (0.0442)	10.59 (0.1240)
20	0.04 (0.0004)	0.03 (0.0004)	0.32 (0.0040)	5.83 (0.0695)	15.14 (0.1743)
25	0.04 (0.0004)	0.04 (0.0004)	0.48 (0.0058)	8.11 (0.0958)	19.20 (0.2182)

Table A.10: Same comparison as in Table A.9, but this time with a truncated Gaussian window.

Brent	0	0.125	0.25	0.375
5	0.06 (1.2567)	0.06 (1.2563)	0.06 (1.2549)	0.06 (1.2506)
10	0.06 (1.2567)	0.06 (1.2552)	0.06 (1.2495)	0.06 (1.2334)
15	0.06 (1.2567)	0.06 (1.2534)	0.06 (1.2411)	0.06 (1.2084)
20	0.06 (1.2567)	0.06 (1.2510)	0.06 (1.2305)	0.06 (1.1806)
25	0.06 (1.2567)	0.06 (1.2482)	0.06 (1.2188)	0.06 (1.1566)

DFT^1	0	0.125	0.25	0.375
5	0.01 (0.1278)	0.01 (0.1821)	0.01 (0.3091)	0.02 (0.5291)
10	0.01 (0.1278)	0.01 (0.2920)	0.03 (0.5860)	0.05 (1.0589)
15	0.01 (0.1278)	0.02 (0.4220)	0.04 (0.8882)	0.07 (1.6346)
20	0.01 (0.1278)	0.03 (0.5654)	0.05 (1.2141)	0.10 (2.2570)
25	0.01 (0.1278)	0.03 (0.7210)	0.07 (1.5638)	0.12 (2.9186)

Table A.11: Error on frequency measured by the Brent and the DFT^1 methods. The analyzed signal is a single sinusoidal oscillator whose frequency remains constant at 2000 Hz while its amplitude is 0.5 modulated by a tremolo. This sound lasts 5 s and $F_s = 44100$ Hz. The average error on frequency in percents (as well as the standard deviation) is given for different tremolo rates and depths. The analysis window is the 512-point Hann window. A boundary indicates when the error is greater than 0.1%.

Brent	0	0.125	0.25	0.375
5	0.00 (0.0027)	0.00 (0.0027)	0.00 (0.0027)	0.00 (0.0027)
10	0.00 (0.0027)	0.00 (0.0027)	0.00 (0.0027)	0.00 (0.0028)
15	0.00 (0.0027)	0.00 (0.0027)	0.00 (0.0028)	0.00 (0.0032)
20	0.00 (0.0027)	0.00 (0.0028)	0.00 (0.0029)	0.00 (0.0052)
25	0.00 (0.0027)	0.00 (0.0028)	0.00 (0.0035)	0.00 (0.0099)

DFT^1	0	0.125	0.25	0.375
5	0.01 (0.1278)	0.01 (0.1817)	0.01 (0.3078)	0.02 (0.5252)
10	0.01 (0.1278)	0.01 (0.2880)	0.03 (0.5747)	0.04 (1.0282)
15	0.01 (0.1278)	0.02 (0.4080)	0.04 (0.8514)	0.07 (1.5393)
20	0.01 (0.1278)	0.02 (0.5331)	0.05 (1.1320)	0.09 (2.0552)
25	0.01 (0.1278)	0.03 (0.6608)	0.06 (1.4152)	0.11 (2.5744)

Table A.12: Same comparison as in Table A.11, but this time with a truncated Gaussian window. The error is always lower than 0.1% for the Brent method.

Brent	0	0.125	0.25	0.375
5	0.78 (0.0039)	0.79 (0.0039)	0.82 (0.0040)	0.89 (0.0042)
10	0.78 (0.0039)	0.81 (0.0039)	0.91 (0.0039)	1.21 (0.0038)
15	0.78 (0.0039)	0.85 (0.0040)	1.07 (0.0044)	1.77 (0.0049)
20	0.78 (0.0039)	0.89 (0.0046)	1.48 (0.0062)	2.91 (0.0081)
25	0.78 (0.0039)	1.08 (0.0057)	2.18 (0.0092)	4.45 (0.0129)

DFT^1	0	0.125	0.25	0.375
5	0.09 (0.0004)	0.07 (0.0005)	0.12 (0.0007)	0.23 (0.0010)
10	0.09 (0.0004)	0.18 (0.0010)	0.39 (0.0019)	0.80 (0.0029)
15	0.09 (0.0004)	0.36 (0.0020)	0.83 (0.0040)	1.73 (0.0059)
20	0.09 (0.0004)	0.63 (0.0034)	1.44 (0.0068)	3.02 (0.0101)
25	0.09 (0.0004)	0.96 (0.0052)	2.22 (0.0103)	4.67 (0.0155)

Table A.13: Error on amplitude measured by the Brent and the DFT^1 methods. The analyzed signal and analysis window are the same as in Table A.11. The average error on amplitude in percents (as well as the standard deviation) is given for different tremolo rates and depths. A boundary indicates when the error is greater than 0.1% (which is always the case here for the Brent method).

Brent	0	0.125	0.25	0.375
5	0.00 (0.0000)	0.02 (0.0001)	0.04 (0.0002)	0.08 (0.0002)
10	0.00 (0.0000)	0.06 (0.0003)	0.14 (0.0007)	0.30 (0.0010)
15	0.00 (0.0000)	0.14 (0.0007)	0.32 (0.0015)	0.68 (0.0022)
20	0.00 (0.0000)	0.24 (0.0013)	0.57 (0.0026)	1.20 (0.0039)
25	0.00 (0.0000)	0.38 (0.0020)	0.88 (0.0040)	1.87 (0.0061)

DFT[1]	0	0.125	0.25	0.375
5	0.04 (0.0002)	0.03 (0.0002)	0.05 (0.0003)	0.10 (0.0004)
10	0.04 (0.0002)	0.07 (0.0004)	0.16 (0.0008)	0.33 (0.0012)
15	0.04 (0.0002)	0.15 (0.0008)	0.34 (0.0016)	0.72 (0.0025)
20	0.04 (0.0002)	0.26 (0.0014)	0.60 (0.0028)	1.26 (0.0042)
25	0.04 (0.0002)	0.40 (0.0021)	0.92 (0.0043)	1.94 (0.0064)

Table A.14: Same comparison as in Table A.13, but this time with a truncated Gaussian window.

Sound Models for Computer Music
– analysis, transformation, synthesis of musical sound –

Abstract

Sounds are physical phenomena belonging to the physical world. In order to manipulate digital sounds using a computer, we need a sound model, that is a formal representation for audio signals. Sound modeling draws the link between the real – analogical – and mathematical – digital – worlds.

Spectral models based on additive synthesis provide general representations for sound well-suited for expressive musical transformations. We introduce the Structured Additive Synthesis (SAS) model which imposes constraints on the additive parameters in order to make these transformations both computationally efficient and musically intuitive, in accordance to perception and musical terminology. This model breaks the arbitrary boundary between sound and music. Its applications in the fields of creation and education are numerous.

A new analysis method is proposed in order to accurately extract the spectral parameters for the model from existing sounds. This method extends the classic short-time Fourier analysis by also considering the derivatives of the sound signal, and it can work with very short analysis windows. Moreover, the reanalysis of the spectral parameters turns out to be extremely useful for difficult problems such as lossless compression or source separation for example.

Finally, a very efficient synthesis algorithm – based on a recursive description of the sine function – can reproduce sound in real time from the model parameters. This algorithm allows an extremely fine control of the partials of the sounds while avoiding signal discontinuities as well as numerical imprecision, and with a nearly optimal number of operations per partial. We consider psychoacoustic phenomena such as masking in order to reduce on the fly the number of partials to be synthesized.

We have also developed the *InSpect*, *ProSpect*, and *ReSpect* free software programs for the purposes of sound analysis, transformation, and synthesis, respectively.

Discipline: Computer Science

Keywords:
computer music,
sound modeling,
sound analysis and synthesis,
enhanced Fourier analysis,
real-time additive synthesis.

Résumé

Une représentation mathématique est nécessaire à la manipulation informatique des phénomènes physiques que sont les sons. La modélisation sonore jette un pont entre le monde réel, physique, analogique et le monde formel, mathématique, numérique.

Les modèles spectraux basés sur la synthèse additive fournissent des représentations formelles pour les sons bien adaptées à leur manipulation informatique (performance) et musicale (expressivité). Cette thèse propose le modèle de Synthèse Additive Structurée (SAS) qui contraint les paramètres additifs pour une manipulation efficace et plus intuitive, en accord avec la perception et le vocabulaire musical. Ce modèle brise la frontière arbitraire entre musique et son, et ses applications à la création et à la pédagogie sont nombreuses.

Une nouvelle méthode d'analyse sonore est proposée afin d'extraire avec une grande précision les paramètres spectraux du modèle à partir de sons naturels. Cette méthode étend l'analyse de Fourier à court terme classique en tirant parti des dérivées mathématiques du signal sonore et nécessite en pratique de petites fenêtres d'analyse. De plus, la réanalyse des paramètres spectraux se révèle être utile pour des problèmes difficiles comme la compression sans perte ou la séparation de sources par exemple.

Enfin, un algorithme de synthèse très efficace, basé sur une description récursive de la fonction sinus, permet de reproduire en temps réel les sons à partir de leur modélisation. Cet algorithme autorise le contrôle précis des partiels des sons tout en évitant les discontinuités du signal et les instabilités numériques, avec un nombre d'opérations par partiel quasi-optimum. La réduction à la volée du nombre de partiels à synthétiser, en tirant parti de phénomènes psychoacoustiques comme le masquage, est aussi envisagée.

Les logiciels libres *InSpect*, *ProSpect* et *ReSpect* sont également proposés pour, respectivement, l'analyse, la transformation et la synthèse sonore.

Discipline : INFORMATIQUE

Mots-clés :
informatique musicale,
modélisation sonore,
analyse et synthèse du son,
analyse de Fourier améliorée,
synthèse additive en temps réel.

U.F.R. DE MATHÉMATIQUES ET D'INFORMATIQUE

LaBRI
Université Bordeaux 1
351, cours de la Libération
F-33405 Talence cedex, FRANCE