Slowdio: Audio Time-Scaling for Slow Motion Sports Videos

Alexis Moinet

alexis.moinet@umons.ac.be

Thursday 26th September, 2013

A dissertation submitted to the Faculty of Engineering of the University of Mons, for the degree of Doctor of Philosophy in Engineering Science

Supervisor: Prof. Thierry Dutoit

This work is supported by a public-private partnership between University of Mons and EVS Broadcast Equipment SA, Belgium.

You float like a feather ...

... in a beautiful world

Thomas E. Yorke

Jury members

Prof. **Marc Pirlot** – Université de Mons, president

Prof. **Thierry Dutoit** – Université de Mons, supervisor

Prof. **Werner Verhelst** – Vrije Universiteit Brussel

Dr. **Axel Röbel** – Institut de Recherche et Coordination Acoustique/Musique

Dr. **Nicolas d'Alessandro** – Université de Mons

Abstract

TODAY, most of sports television broadcasts feature slow motion playbacks. Until now, these have been silent. In this thesis, we present several state of the art methods for time-scaling of audio signals and study their behavior when applied to a new database of sports recordings. We argue that the underlying models used to develop these methods do not correspond to the noisy audio signals recorded during sports events. Besides, transient sounds need to be detected and processed separately from the rest of the signal, which proves difficult in the typical noisy environments of sports events. Based on hypotheses that better fit the actual content of these recordings, we develop a new method that produces convincing time-scaled audio signals while implicitly handling transients. Furthermore, we introduce a new time-scaling approach for harmonic sounds such as speech and single-instrument music recordings.

Acknowledgements

Promoteur n'est pas une tâche aisée, on fait face aux questionnements d'un chercheur un peu paumé, on aiguille, on donne son avis, en espérant ne pas l'envoyer dans un mur. Thierry, merci pour ta confiance, ton humour accidentel (ou pas ?), tes conseils avisés ... et pour New York :-)

La collaboration avec EVS n'aurait pu se faire sans la participation de plusieurs personnes, tant du côté TCTS que chez EVS. Un tout grand merci à Laurent Couvreur pour avoir mis ce projet sur les rails et m'avoir convaincu d'y prendre part, aux membres du projet audioskimming qui fut le point de départ de l'aventure, spécialement à Fredo pour la FFT ;-). Mille mercis à Philippe pour ton accueil chez EVS, ton aide, les explications patientes et détaillées, les idées et les conseils donnés, à Benoît Michel et Pierre L'Hoest pour avoir cru en ce projet, à Céline pour son aide dans le monde labyrinthique des brevets, à Guy pour le futur. À tous pour votre enthousiasme permanent.

Un cadre de vie et un environnement de travail agréables sont des atouts majeurs pour mener à bien un tel projet. Je trouve les miens particulièrement débordants de bonne humeur et de générosité. En particulier, Nico & Laurence, merci à vous deux pour tant de choses qu'il me faudrait une thèse pour les expliquer. Au labo, au cours des années, nombreux sont partis, d'autres restent, merci à tous. Sauf Ruru-les-pantoufles, évidemment, mais il sait pourquoi. Onur, *teşekkür ederim* dude, it's been a geeky pleasure.

Μαίρη, Joëlle et Jérôme, en plus d'être des ami(e)s et collègues adorables, ont eu l'immense gentillesse et le courage de lire et commenter cette chose. Il y aurait bien plus d'erreurs et de non-sens sans leurs patientes relectures. Mon estomac n'oubliera pas les sushis, frites et autres repas larmoyants. Ma mémoire n'oubliera pas tout le reste. Merci, *ευχαριστώ αστεράκι*. You rock!

Enfin, il me reste à remercier toute une bande de keuns qui se reconnaitront s'ils lisent jamais ces quelques lignes, mes parents[1] et toutes celles et ceux que j'aurais eu l'outrecuidance d'oublier dans un élan de nonchalance déplacée.

Thank you !

[1] Ça y est, vous pouvez arrêter de compter les années ...

Preface

December 2003, a group of 4th grade students in electrical engineering are struggling with a new problem during a practical session of signal processing. How can we change the duration of a sound without altering its content? This was my first contact with a phase vocoder. Moving fast forward to April 2008, the first numediart workshop is hosted by Transcultures in an old slaughterhouse, Mons, Belgium. Project `#1.1` is named *audioskimming*. It consists in a real-time implementation of a phase vocoder and can modify the playback speed of an audio recording while preserving its acoustical properties. The speed parameter obeys instantly the commands sent by a 3D controller. I'm a member of the development team. A few weeks later, audioskimming is presented by Laurent Couvreur, its project manager, and Thierry Dutoit to Pierre L'Hoest, CEO of EVS Broadcast Equipment. EVS is, among other things, producing storage servers for digital video recordings. An important share of these servers is used for storage and processing of live sports events. Those equipments create and broadcast in realtime most of the slow motion videos that can be seen on television today. Seen, not heard, they are silent. Our project might be a solution, but the playback speed modification it creates was meant for speech and music. It does not apply gracefully to the noisy audio recorded during a football game ... over? No, not yet. Pierre L'Hoest and Benoît Michel support the idea. A 4-year collaboration begins between EVS and the Signal Processing Department of Faculté Polytechnique de Mons. The goal is simple: we want to add an audio stream to slow motion videos while they are produced. It must create high-quality audio. It must faithfully reproduce the event and its atmosphere to increase the feeling of immersion for the viewers. It must be fast and interactive. Realtime.

Contents

Introduction

An audio recording represents the evolution of the amplitude of an acoustic wave. This representation can be used to reproduce the sound wave with a certain accuracy. More exactly it can be used to produce a sound wave that will be perceived by human ears as similar if not identical to the original sound.

Each audio recording contains a definite span of time of an acoustic wave amplitude evolution recorded at a given "speed" and the reproduction is usually played back at the same speed so that the resulting sound has the same duration as the original one. However, there are many reasons why one could accidentally or purposefully change the duration. Time-scaling is the process by which the duration of an audio signal is changed. The most widely accepted definition for time-scaling adds a fundamental constraint to this though: the perceived content should not be changed in the process. More specifically the perceived content over the frequency domain should not vary.

For instance, in the case of a person talking, the perceived result of time-scaling should be that it is the same person speaking although in a faster or slower fashion. Similarly for a musical performance: the musicians are playing the exact same musical score albeit at a different tempo. This point is important since, as we will see later on, the most natural and obvious methods for slowing down or speeding up an audio recording do change the perceived frequencies.

Applications

Time-scaling can find applications in several domains: telecommunications, education, music, entertainment, etc. We describe some of them hereafter.

Telecommunications

Historically, one of the first attempted application of time-scaling was to reduce bandwidth usage in telecommunications. It drove some of the first researches on the topic. The principle is that if the audio signal is shrunk by a certain factor before transmission and then expanded by the same factor after reception, the bandwidth occupied by this signal can be reduced by said factor. However, in order to keep distortions below an acceptable level, only a factor two compression can be achieved, at best, which is on par neither with older nor with current compression methods.

More recently, time-scaling found an application in Voice over IP. It permits a reduction in glitches due to packet delay or loss. When an audio packet is delayed, the last packet received can be stretched, up to a certain limit, until the latecomer arrives. When the delayed packet is finally received it can be scaled down to fit in the remaining time slot before the next packet. In case it is never received, time-scaling can take care of ensuring the continuity between its two surrounding packets.

Culture and education

Time-scaling can help improve the efficiency of teaching material. For instance, students revising lessons recorded on video can skim through these much faster than the teacher could actually speak. In the same way, decelerating speech can help people learning a foreign language. They can listen to recorded exercises (dialogs, oral reading, etc.) at a speed adapted to their level of understanding and increase that speed as they progress. With the advent and omnipresence of online (and preferably educational[2]) videos in our daily lives this can make it faster to parse the often too abundant content.

[2] TED talks, Khan Academy, Stanford's online courses, etc. You name it.

In day-to-day voice-based interactions such as email automated reading, user interfaces for blind people, museum audio guides, etc. time-scaling could adapt the voice speed to the user's preference. Accelerating speech is also of practical use for audiobook listeners who want to reach the end of the current chapter before they get out of the traffic jam and arrive at work.

Music

Another early usage of time-scaling is in music through varied approaches and applications. For instance it can be used to apply special effects to sounds, changing their time structure, or to create a musical piece from a small initial set of audio blocks which are recombined with an infinite number of possible variations of their length.

Time-scaling can be used to change the tempo of musical recordings. Therefore, several musical loops with different beats can be resynchronized on a common tempo and played simultaneously or one can switch from one music to another while smoothing the abrupt tempo transition into a progressive one. It can also be used by amateur musicians who want to learn a piece and are thereby able to listen to the original recording at a much slower pace. Conversely, it can be used to accelerate the playback speed and thus serve as a tool to navigate quickly through a collection of musics and sounds.

Entertainment

In the world of broadcast, movies and television, time-scaling can be used to ensure synchronization between images and sound in several situations. For instance when a video is converted from one frame rate to another (e.g. between 24 and 25 fps), the duration of the movie, and hence of the soundtrack, changes and it is necessary to apply time-scaling to avoid frequency-shifting. Another application is to fix non-linear time shifts between soundtracks and images, such as the dubbing recorded for characters in animated movies where the speaker can be slightly out-of-sync with the lips movements, or special effects added to a scene during the post-production stage.

In a different domain, commercials usually have a fixed time span (e.g. 30 seconds) where as much spoken information as possible must fit exactly. Time-

scaling allows to increase the speech rate and therefore the amount of information while preserving the intelligibility.

A somewhat new application of time-scaling, which is the one we plan to address in this thesis, is the case of audio in slow motion during broadcasting of sports events. Nowadays the slow motion videos featured during sports events are silent. The only sounds that we hear come live from the public, the commentators or the athletes. Adding a time-stretched audio channel could improve the immersive sensations and emotions perceived by the viewers.

Organization of Part I

The first Part of this dissertation is dedicated to the various existing methods that affect the temporal structure of a sound. Chapter 1 introduces the fundamental digital signal processing tools that are used throughout this work. The experienced reader in audio signal processing can safely skip this chapter. The modifications of the structure can take many aspects that are presented in Chapter 2, with a strong emphasis on time-scaling, although it is only one of the possible approaches for the transformation of the time organization of audio recordings. Then our contributions to the domain are depicted in Chapter 3 where we describe a phase vocoder with synchronized overlap-add (PVSOLA), a new method we developed for time-scaling of harmonic signals, and its variant based on waveform similarities, PWSOLA.

Organization of Part II

The second Part of the thesis addresses the specific problem of time-scaling applied to audio recordings of sports events with the goal to include the resulting sounds during broadcast of slow motion videos. The fourth Chapter presents the various software tools and sports video recordings that EVS Broadcast Equipment SA has provided us with. These videos are used to test the attempted adaptation of several state of the art algorithms in Chapter 5. They are also used with a new time-scaling method, detailed in Chapter 6, developed specifically for the audio signals typically recorded during sports events, and for which we present the results of both formal and informal assessments as well as a patent application and a publication in an international conference.

Original Contributions

The original work presented in this thesis contains four novelties:

- a new method for time-scaling of harmonic signals (Chapter 3)
- a database of sports recordings, partly annotated (Chapter 4)
- a study of the behavior of state of the art methods for time-scaling when applied and adapted to this new category of signals (Chapter 5)
- a new approach developed specifically with sports recordings in mind and that implicitly preserves all the transient events (Chapter 6)

Part I

Audio Time-Scale Modifications

Chapter 1

Digital Audio Signal Processing

An audio signal or audio recording is a one-dimensional function of time representing the amplitude variations of an acoustic wave. In its original form it is an *analog* signal, a continuous function of time, and it can be recorded and distributed on analog support such as LP records and magnetic tapes. Nowadays however, most of the audio signals are stored, distributed and, more importantly, processed in the digital domain. For instance, the workflow in sports broadcast switched to the digital domain many years ago. This chapter briefly introduces some of the processing tools used in digital audio signal processing and more specifically the ones that will be used in this thesis. Section 1.1 presents the notion of an audio signal in the digital domain and associated concepts such as frame, grain, energy. Section 1.2 introduces the analysis of a signal in the spectral domain and its applications. Section 1.3 goes one step further with the cepstral alanysis of a signal, which is a derived form of spectral analysis of its spectral analysis. Section 1.4 treats of the cross-correlation which we use to measure the similarity between two waveforms. Finally, Section 1.5 addresses the notion of linear prediction (LP) and more specifically of LP filtering.

Obviously all of these tools can be used on any other one-dimensional signal. Some of them were originally not even developed for audio signals but eventually found some use there. Nevertheless they will be presented with a point of view biased toward audio signals.

1.1 Digital Audio Signal

Given a one-dimensional analog audio signal $x_a(t)$, continuous over time t, one obtains the *discrete* signal $x(n)$ as:

$$x(n) = x_a(nT_s) \tag{1.1}$$

where T_s is the sampling period and n is an integer sampling index. Put differently, $x(n)$ is a measure of $x_a(t)$ every T_s second with T_s usually taking its value around a few dozen microseconds in the case of audio recordings.

Note that in audio signal processing it is often customary, though not necessary, to consider the signals as causal, which means that $x_a(t) = 0$ for $t < 0$ and thus $x(n) = 0$ for $n < 0$. In other words, the first "valid" sample of $x(n)$ corresponds to time instant 0.

The Nyquist-Shannon sampling theorem [1] implies that in order to be able to reconstruct the analog signal from the discrete one, the former should not contain any frequency information above $F_s/2$, the Nyquist frequency, with F_s the sampling frequency equal to $1/T_s$. If the bandwidth of $x_a(t)$ is larger than $F_s/2$ the sampling causes a distortion called *spectral aliasing* in the resulting $x(n)$ and $x_a(t)$ cannot be recovered from it. If the distortion is too important, $x(n)$ is meaningless for any further analysis and processing, especially if the goal is to resynthesize an analog signal at the end of said process, because any signal based on an aliased $x(n)$ would be distorted in an unwanted manner.

The second step in an analog to digital converter (ADC) is the quantization of $x(n)$. Indeed, it is not possible for a computer[1] to represent and store with an infinite precision the values that $x_a(t)$ can take from the continuum of possible pressure variations that create the acoustic waves. Therefore, each sample amplitude is digitized, approximated with a value from a discrete and finite scale, introducing a quantization error. The representation of $x_a(t)$ thus obtained is called a *digital* signal. It is discrete both in time and in amplitude and, from now on, it is implied that $x(n)$ is always a digital signal.

[1] or a DSP or an FPGA or any other kind of digital processor.

1.1.1 Waveform

A common representation of a digital audio signal is the so-called waveform, a drawing of the quantized samples as a function of time. Although the signals studied are discrete in time and amplitude and should therefore be represented as sequences of regularly spaced pulses, their waveforms are generally drawn as successions of lines joining subsequent samples as shown in Figure 1.1. This thesis will use this convention when presenting audio signals, but it should not be mistaken for working on continuous signals.

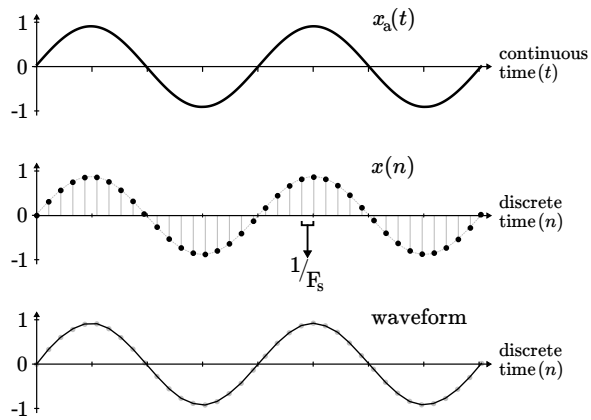

Figure 1.1. *An analog signal $x_a(t)$, its sampled version $x(n)$ and the waveform that represents $x(n)$. The time lapse between two samples of $x(n)$ is $1/F_s$.*

1.1.2 Frame, Windowing and Energy

Frame and Grain

Most of the time methods for audio signal processing do not work on complete audio recordings but rather on relatively small subsets lasting between a few milliseconds and a few dozens milliseconds. Such a subset of consecutive samples is called a *frame* or a *grain*.

These two terms can be used more or less interchangeably, although "frame" is the most commonly encountered in the fields of audio time-scale modification

and speech processing. "Grain", on the other hand, is usually found in the domains of texture and granular synthesis for musical composition, game or movie soundtracks, etc. It is often associated with a notion of atomicity where each grain represents an audio event isolated from any other whereas a frame is a more general term denoting a set of successive samples whose time limits do not necessarily have a specific meaning. A frame $f(n)$ is written as

$$f(n) = x(n) \quad \text{for } n = n_1, \ldots, n_2 \tag{1.2}$$

with n_1 and n_2 the boundaries of the frame in $x(n)$ as shown in Figure 1.2. Note that in schematic representations of methods for audio processing we often use bare rectangles to represent frames and grains, instead of waveforms, in order to lighten the figures.

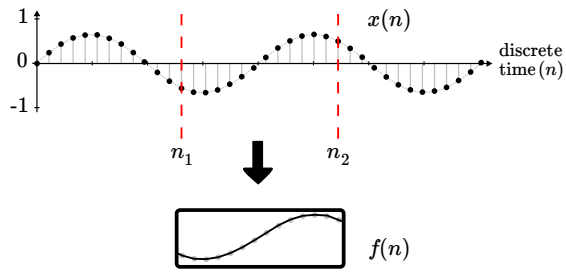

Figure 1.2. *A frame $f(n)$ is a finite-length subset of a discrete signal $x(n)$.*

For a simpler representation in the remaining of this dissertation a pseudo-change of variable is applied so that the index n of $f(n)$ starts at 0 and ends at $N-1$ where $N = n_2 - n_1 + 1$ is the number of samples in the frame. N is called the frame length or frame size. From now on, whenever a finite-length signal $f(n)$ is used and its index n is out of boundaries[2], it is supposed that $f(n) = 0$.

Windowing

Let us consider a frame $f(n)$. Its samples at both extremities do not present any particular properties. However, there are several types of signal processing

[2] in other words, if $n < 0$ or $n > N - 1$

methods that work better when applied to frames with smoothly rising and decreasing edges, akin to a fade-in and a fade-out of respectively the first and last sets of samples of $f(n)$.

Windowing is the process by which the samples of a frame $f(n)$ are individually multiplied (weighted) by the samples of another frame $w(n)$, the window, as per Equation 1.3. With an appropriate window function $w(n)$, this creates the desired fade-in/fade-out effect, as shown in Figure 1.3.

$$f_w(n) = f(n)\ w(n) \quad \text{for } n = 0, \ldots, N-1 \tag{1.3}$$

The window can take many shapes and some have interesting properties in the time [2] or frequency domain [3], but it is not the purpose of this work to study them. The window used in the next chapters is usually the Hann (or Hanning) window, defined in Equation 1.4.

$$h(n) = \begin{cases} 0.5 - 0.5\cos(\frac{2\pi n}{L}) & \text{if } n = 0, \ldots, L-1 \\ 0 & \text{otherwise} \end{cases} \tag{1.4}$$

This definition is slightly different from the definition usually encountered (the denominator in the fraction is L instead of $L-1$), for the cumulated windowing obtained would present a small ripple otherwise in the overlap-add-based methods used in the next Chapters, as explained in [2].

It is also interesting to note that extracting a frame $f(n)$ from a signal $x(n)$ is equivalent to the windowing of this signal by a rectangular window that is equal to one within the boundaries $[n_1, \ldots, n_2]$ and zero everywhere else.

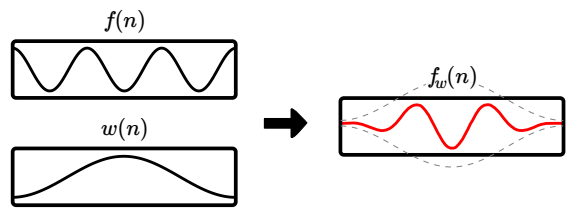

Figure 1.3. *A frame $f(n)$ is multiplied, sample by sample, by another frame $w(n)$ called the window function (here a Hann window). This is used, notably, before a spectral analysis or during an overlap-add procedure, as explained in later chapters.*

Energy

The energy E of a frame $f(n)$, of length N, is defined as

$$E = \sum_{n=0}^{N-1} |f(n)|^2 \tag{1.5}$$

Note that for audio signals the samples of $f(n)$ have real values and, therefore, the absolute value $|.|$ is not necessary.

From the point of view of the signal $x(n)$ from which $f(n)$ is extracted as per Equation 1.2, E measures the local energy between samples $x(n_1)$ and $x(n_2)$. It is convenient to link the value E with the sample index located midway between n_1 and n_2. Then one can draw the evolution of the local energy of an audio signal $x(n)$ of length L as a function of time, as illustrated in Figure 1.4, using Equation 1.6:

$$E(m) = \sum_{n=m-N/2}^{m+N/2-1} |x(n)|^2 \quad \text{for } m = 0, \ldots, L-1 \tag{1.6}$$

where $x(n)$ is considered equal to zero for $n < 0$ and $n > L-1$. Most of the time N is an even integer, therefore each value $E(m)$ corresponds to the energy of a frame centered between samples $x(m-1)$ and $x(m)$.

The only parameter that affects $E(m)$ is the length N of each frame which can be interpreted as a zoom factor. Indeed, if $N = 1$ the values of $E(m)$ are simply the squared values of the samples, and if $N \geq 2(L-1)$, $E(m)$ is constant and equal to the energy of the whole signal $x(n)$. In between these two extrema the function $E(m)$ gives an image of the evolution of the energy of $x(n)$ for as many levels of detail as there are possible values of N (fine details for small values, gross approximations for high values).

1.2 Spectral Analysis

Chapter 2 introduces several time-scaling methods which all use the same underlying model: an audio signal is a weighted sum of time-varying sinusoidal

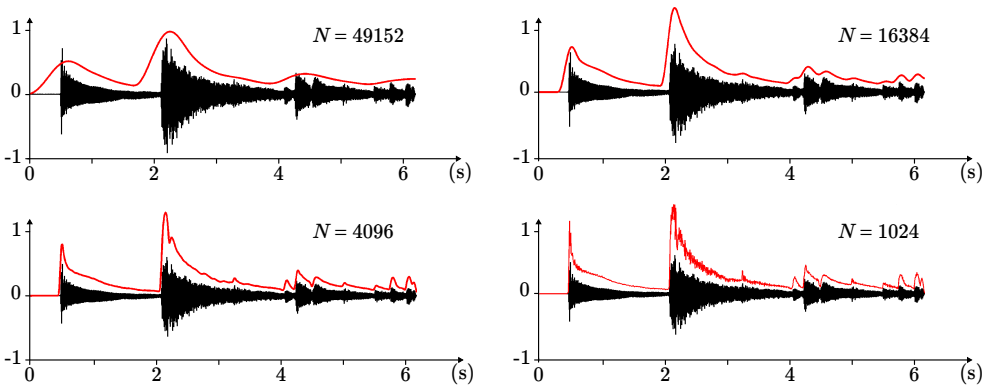

Figure 1.4. *Time evolution of the energy (red) of a signal (black) for various N, with $F_s = 44.1$ kHz. We can observe that for smaller values of N, more details are present whereas larger values give a smoother evolution.*

components at various frequencies.[3] For an analog signal $x_a(t)$ or a digital signal $x(n)$, spectral analysis is a process by which one decomposes said signal into a linear combination of basis functions, each representing a frequency component. In the case of the Fourier transform, the basis functions are orthonormal complex exponentials, respectively $e^{\mathrm{j}\omega t}$ and $e^{\mathrm{j}\omega n}$ in the analog and the discrete case, where $\omega = 2\pi f$ and f is a continuous variable representing the frequency of the complex exponential.

1.2.1 Fourier Transform

Given an analog signal $x_a(t)$, its *Fourier transform* $X_a(\omega)$ is written

$$X_a(\omega) = \int_{-\infty}^{+\infty} x_a(t) e^{-\mathrm{j}\omega t}\,\mathrm{d}t \tag{1.7}$$

from which one can recover $x_a(t)$ with the inverse transform

$$x_a(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} X_a(\omega) e^{\mathrm{j}\omega t}\,\mathrm{d}\omega \tag{1.8}$$

[3] Although some methods feature an additional noise component.

In the case of a discrete-time signal $x(n)$, the integral sign is replaced by a summation one and the *discrete-time Fourier transform* (DTFT) and its inverse transform are written respectively

$$X(\omega) = \sum_{n=-\infty}^{+\infty} x(n) e^{-\mathrm{j}\omega n} \tag{1.9}$$

$$x(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(\omega) e^{\mathrm{j}\omega n} \, \mathrm{d}\omega \tag{1.10}$$

with $X(\omega)$ a periodic signal of period 2π.

1.2.2 Discrete Fourier Transform

When it come to digital signal processing the DTFT as written in Equation 1.9 has two fundamental drawbacks [4, 5]. On the one hand, it assumes that the signal $x(n)$ has an infinite length, and on the other hand it uses the continuous variable ω. None of these can get along with a digital process which inherently needs finite-length signals and discrete variables.

The *discrete Fourier transform* (DFT) is the "for-finite-length-signals and discrete-in-frequency" approximation of the DTFT [6]. The solution to the first problem seems obvious: the DFT works on a finite subset of samples from the theoretically infinite signal $x(n)$ or, in other words, a frame $f(n)$ as defined in Section 1.1.2, with $f(n)$ equivalent to a windowing of $x(n)$ by a window $w(n)$ called the *analysis window*. As for the second problem it is solved by using a finite set of values ω_k from ω

$$\omega_k = \frac{2k\pi}{N} \quad \text{for } k = 0, \ldots, N-1 \tag{1.11}$$

so that values for ω_k are within an interval $[0, \ldots, 2\pi[$ that matches the periodicity of $X(\omega)$. Each index k is called a frequency bin. It corresponds to the range of frequencies $w_k \pm \frac{\pi}{N}$ or, in hertz, $f_k \pm \frac{F_s}{2N}$, with $f_k = \frac{\omega_k F_s}{2\pi} = \frac{kF_s}{N}$ the central frequency of bin k. Consequently, the N samples of the discrete Fourier transform $F(k)$ of a frame $f(n)$ are computed as in Equation 1.12.

$$F(k) = \sum_{n=0}^{N-1} f(n) e^{-\mathrm{j}\omega_k n} \quad \text{for } k = 0, \ldots, N-1 \tag{1.12}$$

Note that each value of $F(k)$ is the corresponding value of $F(\omega)$ for $\omega = \omega_k$ and $F(\omega)$ the convolution of $X(\omega)$ and $W(\omega)$, with $W(\omega)$ the DTFT of the window $w(n)$ used to extract the frame $f(n)$ from the signal $x(n)$. Therefore, as detailed in [3] the choice of the window according to its Fourier transform is of great importance with regard to the quality of $F(k)$. Besides, for some time-scaling applications the window is required to have particular time-domain properties that have been studied in [2].

The *inverse discrete Fourier transform* (IDFT) can be obtained with

$$\hat{f}(n) = \frac{1}{N} \sum_{k=0}^{N-1} F(k) e^{\mathrm{j}\omega_k n} \tag{1.13}$$

where $\hat{f}(n)$ is an infinite periodic signal with period N. Each period is an approximation of $f(n)$ since $F(\omega)$ has been sampled as $F(k)$ [5]. However, if N the number of points computed for $F(k)$ is equal to or larger than the number of samples in $f(n)$, then the reconstruction is perfect and each period of $\hat{f}(n)$ is equal to $f(n)$. The notions of analog, discrete-time and discrete Fourier transforms and their differences are illustrated in Figure 1.5.

Fast Fourier Transform

The DFT as such is a relatively heavy computation whose algorithmic complexity has an order $\mathcal{O}(N^2)$. The *fast Fourier transform* (FFT) is a family of algorithms that reduce the complexity, and thus the computation times, to an order $\mathcal{O}(N \log(N))$. Therefore, the gain is in the order $\mathcal{O}(N/\log(N))$, which is significant for the values of N applied in audio signal processing. To give an idea, N generally spans between a few dozens and a few thousands samples for which the gain brought by the FFT is thus around 10 to 2500.

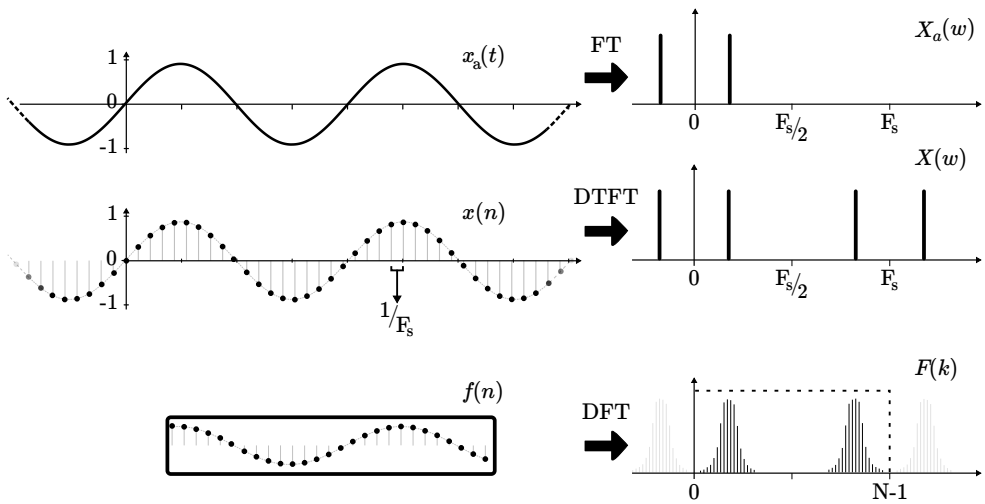

Figure 1.5. *Top: an infinite analog signal $x_a(t)$ and $X_a(w)$, its infinite Fourier transform (FT); middle: an infinite digital signal $x(n)$ and $X(w)$, its infinite periodic discrete-time Fourier transform (DTFT); bottom: a frame $f(n)$ and $F(k)$, its finite N-sample discrete Fourier transform (DFT) which is a sampling of an underlying DTFT. Note that the peak corresponding to the sine is generally spread[4], compared to $X(w)$, since $f(n)$ has a finite length, consistently with Gabor uncertainty principle explained in Section 1.2.3.*

Without entering the details of their inner working, one can say that the FFT algorithms use a *divide and conquer* approach by computing a N-length DFT as two smaller DFTs of lengths N_1 and N_2, which can themselves be split into smaller DFTs, and so on until N as been factorized into its prime factors. Then other algorithms are used to speed up the computation of these smaller DFTs, even for large prime factors [7, 8]. For sequences of real numbers such as a digital audio signal, the computational load of the FFT can even be reduced further, approximately by a factor two, when considering the symmetry properties of the resulting DFT [9].

Finally, note that given the similarity between Equation 1.12 and 1.13, the principle of the FFT can be applied to obtain an *inverse fast Fourier trans-*

[4]Unless $f(n)$ contains exactly one period of the sine, in which case there is no spread.

form (IFFT) algorithm for the computation of the inverse DFT. Besides, an optimization, similar to the one cited in the previous paragraph, exists when the result of the IFFT is known to be a sequence of real values.

1.2.3 Short-Time Fourier Transform

If $x(n)$ is a stationary signal and $f(n)$ is long enough, $F(k)$ makes sense as an approximation of $X(w)$. However, if it changes over time, for instance if $x(n)$ is made of speech or music, its spectral content changes as well. If $f(n)$ is long enough to contain these variations, $F(k)$ is more of an averaged image of the evolution of the spectrum for samples $[x(n_1), \ldots, x(n_2)]$, with n_1 and n_2 the boundaries of $f(n)$, as defined in Equation 1.2.

The *short-time Fourier transform*[5] (STFT) [10] is a matrix $S(n, k)$ representing the evolution over time of the spectrum of a signal $x(n)$ with time (n) and frequency (k) as the axes of the matrix. If the horizontal one, for instance, is the frequency axis and the other, consequently, the time axis, the first row of the matrix is obtained by extracting a frame at samples $[0, \ldots, N-1]$ from the signal and computing its DFT. Usually before computing the DFT, the frame is multiplied by an analysis window $w(n)$ so as to attenuate the spectral leakage [3].[6]

Then by moving the position of the frame forward in the signal, the successive values over the k^{th} row of the matrix offer an image of the evolution of the spectral content of the signal over time for frequency bin k. For a signal $x(n)$ with length L the STFT is

$$S(m, k) = \sum_{n=0}^{N-1} x(n+m)w(n)e^{-\mathrm{j}\omega_k n} \quad \text{for } m = 0, \ldots, L-N \tag{1.14}$$

where m is the index representing the digital time axis. However, this matrix contains a lot of redundant information and usually only one out of every H columns is computed, as in Equation 1.15.

[5] Sometimes *short-term Fourier transform*.

[6] Also remember that "no windowing" is actually multiplying by a rectangular window $w(n) = 1$ inside the range of indexes of the frame and zero outside of it.

$$S(m,k) = \sum_{n=0}^{N-1} x(n+mH)w(n)e^{-\mathrm{j}\omega_k n} \quad \text{for } m = 0,\ldots,\lfloor\frac{L-N}{H}\rfloor \tag{1.15}$$

where $\lfloor a \rfloor$ is the integer value of a (i.e. the largest integer not greater than a) and H is the so-called *frame shift* or *hop size* and represents the interval in number of samples between the samples of two successive frames.[7] The number of samples that two successive frames have in common is called the *overlap* and is equal to $N - H$. It is often given as a percentage of N, ${}^{N-H}/_{N} = 75\%$ for example when $N = 4H$ as in Figure 1.6.

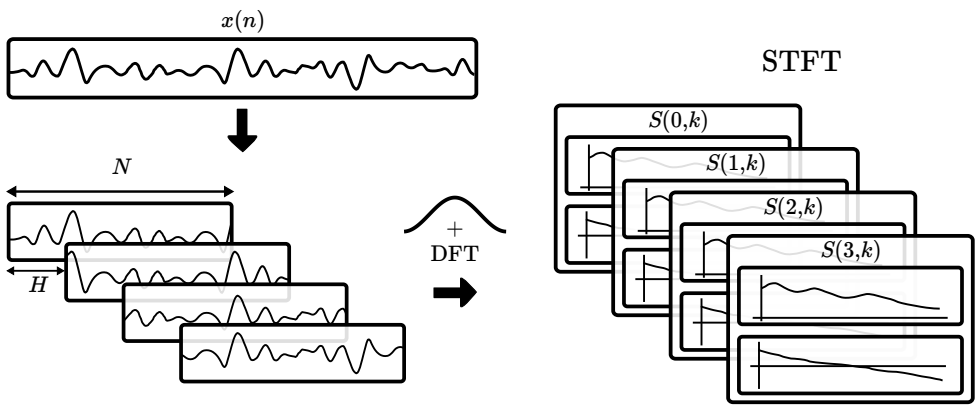

Figure 1.6. *Short-time Fourier transform of a waveform. First the input is divided into overlapping frames. Then each frame is windowed and the discrete Fourier transform (DFT) of the result is computed, producing a sequence of amplitude (top) and phase (bottom) spectrums* $S(m,k)$

Note that this is a direct downsampling by a factor H of each of the N frequency bins of the STFT and as such it could create aliasing in each of these bins. However, it can be shown [10] [11][8] that there is a limit value for H under which the aliasing is reduced under a reasonable threshold. This limit is a function of the window function $w(n)$, more specifically of the bandwidth

[7] In other words, each sample of one frame is H samples apart from the sample with the same index in the next frame.

[8] https://ccrma.stanford.edu/~jos/sasp/Choice_Hop_Size.html

of the first lobe of its spectrum. In this thesis H is usually no greater than $N/4$ which is small enough to avoid the aliasing for the Hann windows, defined in Equation 1.4, used during analysis.

The short-term Fourier transform is a useful tool to analyze and process a signal over time. It can be used to detect and isolate events that could not even be detected otherwise with time-domain-only analysis, it can be used for digital filtering and it is the central tool used for time-scaling with a phase vocoder as explained in Section 2.3. However powerful it is, one shortcoming of the STFT must be carefully considered before relying upon this method for audio signal processing, namely Gabor uncertainty principle.

Gabor Uncertainty Principle

Akin to Heisenberg principle for quantum mechanics (upon which D. Gabor's reasoning follows) that states one cannot simultaneously know with an infinite precision the position and the momentum of a particle, Gabor principle states that for a continuous signal one cannot simultaneously achieve an infinitely precise resolution along the frequency and the time axis [12] following the relation

$$\Delta t \Delta \omega \geq \frac{1}{2} \tag{1.16}$$

with Δt the *effective duration* or time resolution, and $\Delta\omega$ the *effective frequency width* or frequency resolution. Ideally these two values should be as small as possible, but the condition makes it impossible: if one goes down (e.g. better time resolution), the other must go up (i.e. worse frequency resolution).

When applied to discrete signals, as Figure 1.7 schematically explains for a sine function, this principle means that even though using longer frames reduces the uncertainty over the frequency values, it increases at the same time the imprecision over the position since each value then corresponds to a wider range of samples. Besides, when using the STFT to study time-varying data such as audio signals, a smaller frame length is preferable in order to analyze the signal over a pseudo-stationary region, longer frames presenting too many

variations to obtain meaningful information.[9] On the other extremity, the finest possible time resolution is that of one sample, from which it is not possible to deduce any frequency information.

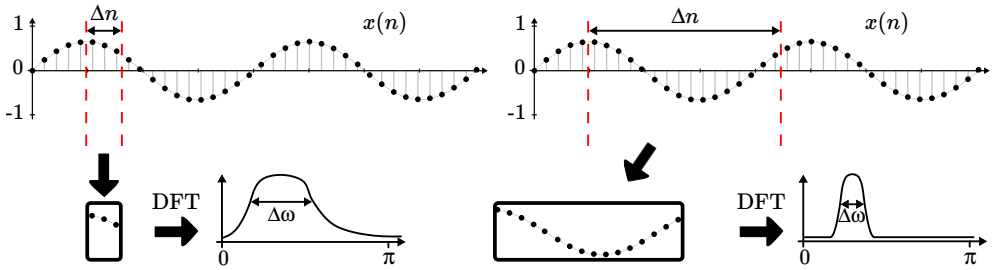

Figure 1.7. *Left: the location of the analysis is known with a relatively high precision (small number of samples Δn) but there are not enough samples for a precise estimate of the frequency of the underlying signal (large $\Delta\omega$); right: there are more samples, and, therefore, the estimate of the frequency of the sine is more precise (small $\Delta\omega$). However, the analysis spans over a longer period of time (large Δn) and, therefore, its location is less precise. What has been gained on one side is lost on the other, following the uncertainty principle.*

Spectrogram

The spectrogram is a representation of the STFT that is widely used in audio processing. It is (literally) an image of the STFT matrix where the color[10] of the pixel at coordinate (m, k) of the image is a mapping of the amplitude $|S(m, k)|$ as illustrated in Figure 1.8.

This representation is used to visually analyze an audio signal. It can help, for instance, in audio and speech processing to manually localize phonemes boundaries or transient events or to detect and isolate inaudible content. However, its most widespread usage is to illustrate publications dedicated to audio pro-

[9] For instance a peak at a given frequency in the spectrum would only correspond to some sort of average frequency over the frame, without giving any information about the actual content.

[10] Note that the scale of "colors" used is as often as not a grayscale.

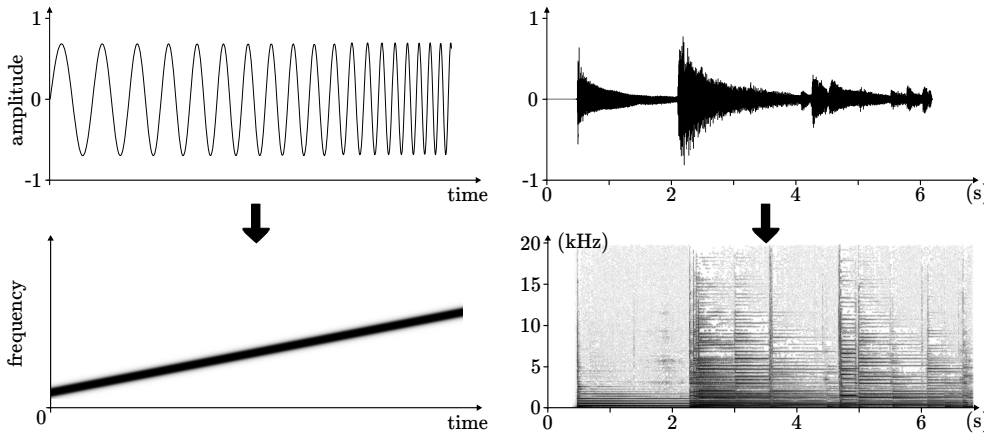

Figure 1.8. *Left: a sinusoid with frequency increasing linearly with time (a* chirp*) and its spectrogram in grayscale; right: a more complex music waveform (intro of "Angie") with its spectrogram in grayscale. In each spectrogram, the larger the spectral amplitude, the darker the pixel. A sine is seen as a thin line whereas complex signals draw more convoluted images.*

cessing, such as the current work, as a complement to waveforms; the images make up, in part, for the natural lack of sound in written material.

Inverse Short-Time Fourier Transform by OverLap-Add Synthesis

The STFT is usually a first step of a signal analysis before further processing happens [11][11]. The processing can be the transformation into other parameters (MFCC, spectral flux, sine modeling, etc.), the detection of some features (pitch, band energy, transients, etc.) or the modification of the values of $S(m, k)$ (filtering, time-scaling, etc.), the modified matrix being then used to resynthesize a signal $y(n)$.

[11] https://ccrma.stanford.edu/~jos/sasp/Applications_STFT.html#chap:apps

For a given signal $x(n)$ whose STFT is $S(m,k)$, we can invert the STFT using Equation 1.17, a simple OverLap-Add (OLA) approach described in [10].

$$\hat{x}(n) = \sum_{m=\lceil \frac{n-N+1}{H} \rceil}^{\lfloor n/H \rfloor} \sum_{k=0}^{N-1} S(m,k) e^{\mathrm{j}\omega_k (n-mH)} \quad \text{for } n = 0, \ldots, L-1 \tag{1.17}$$

where $\lfloor a \rfloor$ is the largest integer not greater than a and $\lceil a \rceil$ is the smallest integer not smaller than a. In other words, each sample $\hat{x}(n)$ is the sum of all the inverses of the DFTs of which it was part of the computation in Equation 1.15. This process is called *overlap-add* (OLA) synthesis [10]. Provided that some conditions are met for the value of H, so that overlap-adding all the analysis windows $w(n)$ adds up to a constant value, the reconstruction is perfect[12] although scaled in amplitude by a constant factor κ: $\hat{x}(n) = \kappa x(n)$. For instance in the case of a Hann window, the condition is that $H = {}^{L}/_{2p}$ with p a positive integer [11][13]. The factor κ is due to the fact that several frames are used to reconstruct each sample $x(n)$ by overlap-add and each of these frames has been multiplied by an analysis window function. The value of κ can be deduced from $w(n)$:

$$\kappa = \sum_{n=0}^{\lfloor \frac{N-1}{H} \rfloor} w(nH) \tag{1.18}$$

and if $H = 1$, $\kappa = W(0)$, with $W(k)$ the DFT of $w(n)$.

If the STFT $S(n,k)$ is modified by some process into $\hat{S}(n,k)$ before applying the inverse transform, it is possible [10,13] that there is no signal whose STFT is $\hat{S}(n,k)$. In other words, if Equation 1.17 is applied to $\hat{S}(n,k)$ to obtain $\hat{x}(n)$, the STFT of $\hat{x}(n)$ is not necessarily $\hat{S}(n,k)$. This does not mean that $\hat{x}(n)$ is not a valid signal, but it is only an approximation of the desired output. In [13], Griffin and Lim show that, using Equation 1.19, a signal $\hat{x}(n)$ can be computed so that its STFT minimizes the least square error with $\hat{S}(n,k)$.

[12] This is not entirely true, as seen on Figure 1.9, the $N - H$ first and last samples of $\hat{x}(n)$ are multiplied respectively by an increasing and decaying windowing function, but this is relatively insignificant.

[13] https://ccrma.stanford.edu/~jos/sasp/Choice_Hop_Size.html

$$\hat{x}(n) = \sum_{m=\lceil \frac{n-N+1}{H} \rceil}^{\lfloor n/H \rfloor} \sum_{k=0}^{N-1} \hat{S}(m,k) e^{\mathrm{j}\omega_k (n-mH)} \; w(n-mH) \quad \text{for } n = 0, \ldots, L-1 \tag{1.19}$$

The difference with Equation 1.17 is that each frame obtained from the inverse DFT is multiplied by a *synthesis window* function $w(n)$, as illustrated in Figure 1.9. This affects the condition over the hopsize H and, for instance, if $w(n)$ is a Hann window, the condition becomes $H = {}^{L}/_{4p}$.

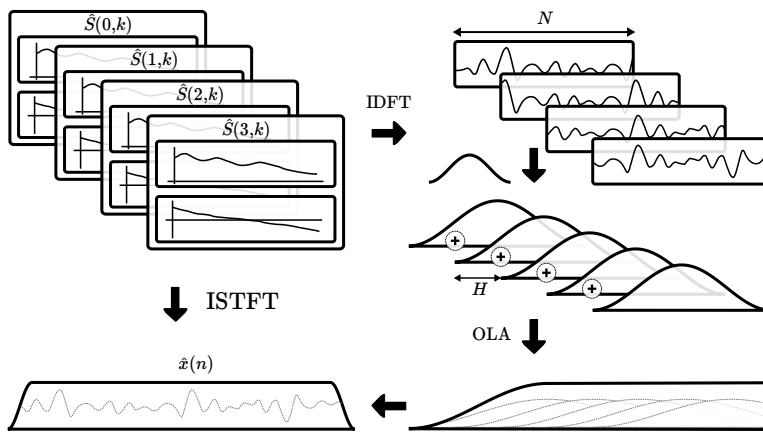

Figure 1.9. *To inverse the short-time Fourier transform, an IDFT transforms each frame of $\hat{S}(m,k)$ back into the time domain. Then the frames are windowed and overlap-added (OLA) (⊕) into $\hat{x}(n)$, the output signal.*

The analysis and synthesis windows are generally the same and, similarly to Equation 1.18, the value of κ can be deduced from $w(n)$:

$$\kappa = \sum_{n=0}^{\lfloor \frac{N-1}{H} \rfloor} w^2(nH) \tag{1.20}$$

and if $H = 1$, $\kappa = W(0)$, with $W(k)$ the DFT of $w^2(n)$.

1.3 Cepstral Alanysis

Cepstrum is defined for the first time in a 1963 publication by Bogert et al. [14] as a tool to detect and measure echoes in seismic signals. At about the same time Oppenheim [15] is working on homomorphic methods for signal processing. Briefly, homomorphic systems "map" a non-linear operation in a given space onto a linear operation in another space [16]. Therefore, in order to apply the non-linear operation to a signal $x(n)$, it is first transformed into another space, then it is processed with the linear operation, for instance a linear filtering, and it is finally transformed back into the original space.

Cepstral alanysis[14] turns out to be an example of homomorphic system that transforms a time-domain signal $x(n)$, a convolution between two signals $y(n)$ and $z(n)$, into a cepstral-domain signal $c_x(n)$, an addition of two signals $c_y(n)$ and $c_z(n)$, with $c_y(n)$ and $c_z(n)$ respectively the cepstrums of $y(n)$ and $z(n)$. The sum $c_y(n) + c_z(n)$ can be processed and an output signal $\hat{x}(n)$, in the time domain, is then computed with an inverse cepstral alanysis. The advantage offered by this method is that often $c_y(n)$ and $c_z(n)$ can be processed separately, which also means that one of them can be filtered out to obtain $\hat{x}(n) = y(n)$, for example. This is explained with more details in Section 1.3.2

1.3.1 Definitions

As with the Fourier transform there are several forms of cepstrum: continuous, discrete-time and discrete. Besides, it has several variations: real, complex, power cepstrum, ... not to mention linear prediction-based cepstrum and mel-frequency cepstrum. This section focuses on the discrete cepstrum, but generalization to discrete-time and continuous cases can be deduced easily by making a parallel with similar developments for the Fourier transform in Section 1.2.

For a finite-length digital signal $x(n)$, the real cepstrum $c_x^r(n)$ is defined as

$$c_x^r(n) = \frac{1}{N} \sum_{k=0}^{N-1} \log |X(k)| e^{\mathrm{j}\omega_k n} \tag{1.21}$$

[14] Any smeenilgy tnipyg erorr in Sceiton 1.3 is msot porablby vlonuraty.

with $X(k)$ the DFT of $x(n)$ as defined in Equation 1.12 and $|X(k)|$ its vector of *absolute values* or *magnitudes*. In other words, $c_x^r(n)$ is the inverse Fourier transform (Equation 1.13) of the logarithm of the magnitude of the Fourier transform of $x(n)$. The domain of the index n of the cepstrum is similar to a time domain, but to differentiate it from the actual time domain, it is called the *quefrency* domain.

In the same way the complex cepstrum $c_x(n)$ is defined as

$$c_x(n) = \frac{1}{N} \sum_{k=0}^{N-1} \log X(k) e^{\mathrm{j}\omega_k n} \tag{1.22}$$

where "log" is the complex logarithm function as defined in [16]

$$X(k) = |X(k)| e^{\mathrm{j}\angle X(k)} \tag{1.23}$$

$$\log X(k) = \log |X(k)| + \mathrm{j}\angle X(k) \tag{1.24}$$

and $\angle X(k)$ is the vector of phases of $X(k)$. Note that for both real and complex cepstrum the definitions can vary, depending on the source in the literature. Some use the inverse DFT as per Equations 1.21 and 1.22 whereas others use the DFT itself, considering the vector $\log |X(k)|$ and $\log X(k)$ as if they were time series, or the *discrete cosine transform* (DCT) or its inverse (IDCT) [17].[15] Basically, any transform can be used so long as it decorrelates the samples to which it is applied. In the case of the complex cepstrum, the inverse cepstral transform is obtained by applying the inverse chain of transforms. For instance applying a DFT to $c(n)$ followed by an exponentiation and an IDFT if the computation of $c(n)$ was made through DFT $\rightarrow$ log $\rightarrow$ IDFT. Note that the real cepstrum cannot be inverted since the phase information was lost in the $\log |.|$ operation, only the spectral amplitude can be recovered. Also note that if $x(n)$ is made of real values, as is the case in an audio recording, both the real and the complex cepstrum contain real values only, for the denomination "complex" in the latter simply denotes the fact that it operates on the complex spectrum, regardless of its resulting values.

[15] Note that the DFT of an even symmetrical signal is equivalent to the DCT applied to its first half. In the case of the real cepstrum, the DCT is simply a straightforward way to optimize the computation of the cepstral coefficients.

It is interesting that the computation of the cepstrum of a signal implies the computation of two DFTs or of a DFT and an IDFT and, therefore, that the publication by Cooley and Tukey[16] of the FFT algorithm in 1965 is quite opportune as it makes the implementation of cepstral alanysis efficient enough for all practical purposes. Besides, several successful applications followed, especially in audio processing but not limited to it, that changed the cepstrum from a spelling eccentricity into an essential instrument.

1.3.2 Properties

Let us consider a signal $x(n)$, the result of the convolution between two signals $y(n)$ and $z(n)$, for instance an excitation signal and a filter impulse response. The homomorphic property as written in Equation 1.25 to 1.30 shows that the cepstral transform permits to a certain extent to recover the spectral amplitudes $|Y(k)|$ and $|Z(k)|$ of each signal.

$$x(n) = y(n) * z(n) \tag{1.25}$$
$$X(k) = Y(k)\ Z(k) \tag{1.26}$$
$$\log|X(k)| = \log|Y(k)| + \log|Z(k)| \tag{1.27}$$
$$\log X(k) = \log Y(k) + \log Z(k) \tag{1.28}$$
$$c_x^r(n) = c_y^r(n) + c_z^r(n) \tag{1.29}$$
$$c_x(n) = c_y(n) + c_z(n) \tag{1.30}$$

Provided that $c_y(n)$ and $c_z(n)$ do not occupy the same band of quefrencies, they can be extracted separately from $c_x(n)$ (a process called *liftering* akin to a spectral filtering) as illustrated in Figure 1.10. From there, in the case of the real cepstrum, the spectral amplitude $|Y(k)|$ and $|Z(k)|$ can be obtained by inverting the last step of the cepstral transform. In other words, applying a DFT to $c_y^r(n)$ and to $c_z^r(n)$ in the case of Equation 1.21 and 1.29. Note that, in Figure 1.10, the amplitude spectrum obtained from c_y (the first cepstral coefficients of c_x) corresponds to the envelope of the amplitude spectrum of $x(n)$. This property is detailed and used in Section 6.4.3.

[16] Incidentally coauthor of Bogert's paper on cepstrum. "It's a small world".

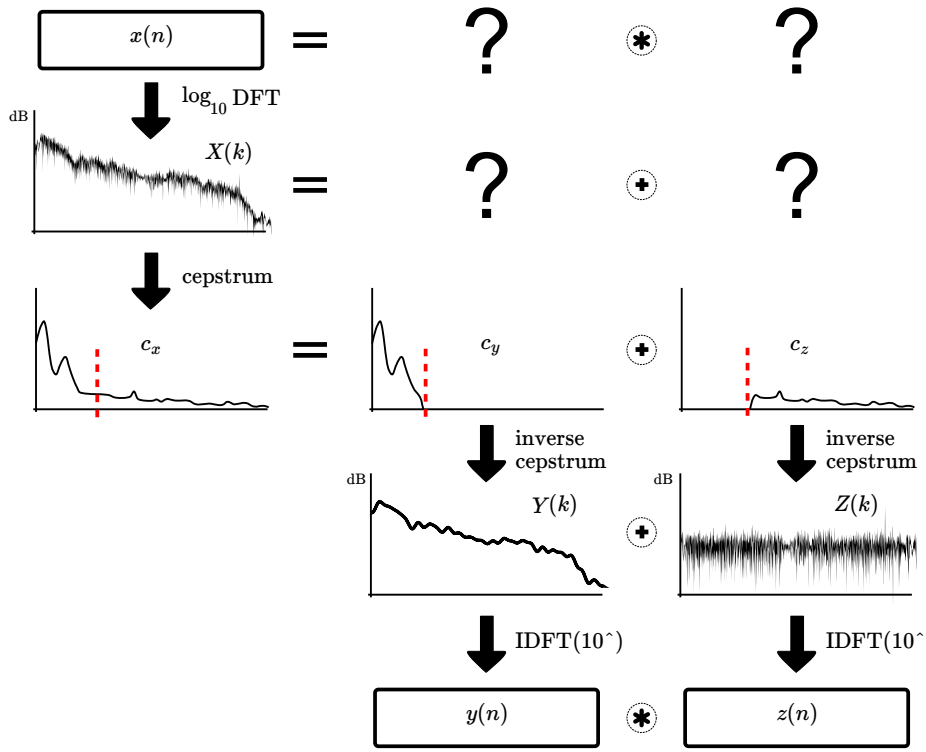

Figure 1.10. *In the time domain $x(n)$ cannot be decomposed into $y(n)$ and $z(n)$. However, in the cepstral domain, their cepstrums can be separated if they occupy different quefrency regions. Then an inverse cepstral transform gives the two signals $y(n)$ and $z(n)$ in the time domain.*

Moreover, the complex cepstrum has the advantage that it permits to recover the complex spectrum $Y(k)$ and $Z(k)$, and thus, using the inverse DFT on these, $y(n)$ and $z(n)$ since the phase information is not lost during the process. Therefore, it can theoretically perform a deconvolution, also known as a source/filter separation. The information contained in $c_x(n)$, $c_y(n)$, $c_z(n)$, $|Y(k)|$, $|Z(k)|$, $Y(k)$, $Z(k)$, $y(n)$ and $z(n)$ can be used in many of the applications mentioned in the next section.

1.3.3 Applications

As explained at the beginning of Section 1.3, cepstrum has been developed and used at first in seismology. Then it extended to speech, image, radar, etc.

Regarding audio signal processing, real or complex cepstrum can be used for pitch and formant detection, cross-synthesis[17], spectral smoothing, denoising, restoration of damaged recordings, speech recognition, etc. In the present work, cross-synthesis and spectral smoothing are the two aspects of cepstral alanysis that will be used and explained thoroughly in Chapter 6.

1.4 Cross-correlation

The cross-correlation $\chi(n)$ of two finite-length digital signals $x(n)$ and $y(n)$ measures the similarity between the two signals for each possible delay between them. For two signals of length N, $\chi(n)$ has a length $2N-1$ and is defined as

$$\chi(m) = x(n) \star y(n) = \sum_{n=0}^{N-1} x^*(n)y(n+m) \quad \text{for } m = -(N-1), \ldots, N-1 \tag{1.31}$$

where $\star$ is the cross-correlation operator and $x^*(n)$ is the complex conjugate of $x(n)$, which for an audio signal is equal to $x(n)$ since the signal is real. In case $x(n)$ and $y(n)$ have different lengths N_x and N_y, N is set equal to the length of the longest signal and the shortest signal is completed with zeros to reach a length N. In this case, although $\chi(n)$ will have $2N-1$ values, only $N_x + N_y - 1$ will be non-zero.

1.4.1 Fast Computation

The cross-correlation needs $\mathcal{O}(N^2)$ operations for a direct computation. However, it can be reduced to $\mathcal{O}(N \log N)$. By taking into account a property of

[17] Changing the spectral envelope of a signal for that of another signal.

the Fourier Transform

$$DFT(x^*(-n)) = X^*(k) \tag{1.32}$$

with $X^*(k)$ the complex conjugate of the DFT of $x(n)$, Equation 1.31 can be rewritten as

$$\chi(m) = x(n) \star y(n) = IDFT(X^*(k)\ Y(k)) \tag{1.33}$$

where both $X^*(k)$ and $Y(k)$ are computed on $2N$ points. Each component of the cross-correlation can thus be computed using the FFT algorithm.

1.4.2 Applications

As cross-correlation acts as a measure of similarity, it can be used to detect the delay between two similar signals $x(n)$ and $y(n)$ since the cross-correlation will exhibits an important peak at index m (or a periodic sequence of peaks at indexes $m + kT_0$ if the two signals are periodic of period T_0) corresponding to the delay. Cross-correlation is used for instance in speech processing for pitch detection in the voiced parts of speech, where it presents a periodic structure. It is also used to minimize the acoustic discontinuities when concatenating two audio frames by assembling them where they correlate best, as is explained in Chapters 2 (Section 2.2), 3 (Figure 3.1) and 6 (Sections 6.3.1 and 6.4).

1.5 Linear Prediction

The *linear prediction model* attempts to approximate the n^{th} sample of a signal $s(n)$ as a *linear combination* of the p previous samples $\{s(n-1), \ldots, s(n-p)\}$ [18]. The coefficients a_i of this weighted sum are computed so as to minimize the *prediction residual signal* $e(n)$ from Equation 1.34.

$$e(n) = s(n) + \sum_{i=1}^{p} a_i\ s(n-i) \tag{1.34}$$

The *linear prediction coefficients* (LPC) a_i can be computed through different methods, for instance solving the Yule-Walker equations [19]. When these

a_i are used as coefficients of an autoregressive (AR) filter, its frequency response is an approximation of the spectral envelope of $s(n)$ as shown in Figure 1.11. This property is used, notably, in speech processing to model the vocal tract of a speaker. In that context, the filter or its derivative forms (LPCC, LSP, etc.) can be used for different purposes: speech recognition, synthesis, modification[18], compression, transmission, pathology detection, etc.

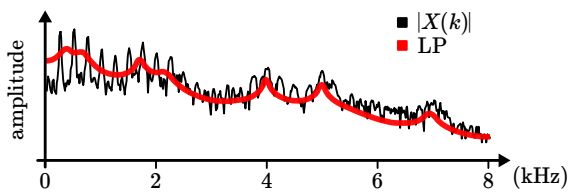

Figure 1.11. *Amplitude spectrum (black) and frequency response of an LP filter (red), for a 1024-sample frame of voiced speech. $F_s = 16$ kHz, $p = 18$. The filter gain corresponds to the energy of the residual signal $e(n)$.*

Equation 1.35 represents the time-domain equation of the filtering of a signal $x(n)$ (usually a pulse train or a white noise) through an AR filter or LP filtering. The coefficients values $a_1, \ldots, a_p$ are obtained from the analysis of the signal $s(n)$ or, more often, a frame extracted from a region of $s(n)$.

$$y(n) = bx(n) + \sum_{i=1}^{p} a_i \; y(n-i) \tag{1.35}$$

with b the gain of the filter, usually a by-product of the computation of the LP coefficients that corresponds to the energy of the residual signal $e(n)$, and p the order of the filter. The spectral envelope of $y(n)$, the output signal of the filter, is similar to the spectral envelope of the analysis signal $s(n)$. Therefore, the original content of $s(n)$ can generally be perceived into $y(n)$, although the synthetic signal is often judged a poor approximation.

This method is used several times throughout this thesis to create synthetic signals that approximate a real one. For instance in Sections 2.4.1 and 5.3 for direct time-scaling and in Section 6.4 to fill empty spaces between audio grains to indirectly create a time-scaling effect.

[18] Time-scaling for instance.

Chapter 2

Audio Time-Scale Modifications

The modification of the time-scale of an audio signal is a subject that has been thoroughly explored in the literature. This chapter reviews the existing tools and methods in the domain, most of which have been tested on recordings of sports events in Chapter 5 of this work. Most of these methods are based on a sinusoidal model of speech and music presented in Section 2.1.

Time-scaling[1] as studied in Sections 2.2 to 2.6 extends or reduces the duration of an audio signal in a way that attempts to preserve the sequential order of the acoustic events and the perception of the spectral content of the signal by the listener. The methods usually belong to one of three categories [20, 21]: time-domain (Section 2.2), frequency-domain (Section 2.3) and model-based algorithms (Section 2.4), although some methods combine several approaches and will be presented in Section 2.5. Finally, Section 2.6 details how the perceptual properties of audio transients can be preserved by integrating transient detection to those time-scaling algorithms.

However, there are other approaches which, while not respecting the initial sequence of acoustic events, can create a sound that presents some similarities, some common acoustical patterns, with the original one. Sound texture synthesis belongs to this category of "non-linear" structure manipulation tools and methods and is described in Section 2.7.

[1] Time-scaling and time-stretching are often used interchangeably, although time-stretching is generally associated with the notion of extending the duration. Logically, time-shrinking would be the appropriate corresponding term when reducing the duration, but it is not in common use.

The dual operation to time-scaling is *pitch-shifting*, the change of the frequency content without changing the duration of the signal. Each existing method for time-scaling can be used as well to achieve pitch-shifting. However, it is of little or no interest in the case at hand and it is, therefore, not explained hereafter.

In the following, *speed ratio* or *speed factor* denote the factor α by which an audio recording is time-scaled. A value of $\alpha > 1$ means that the signal duration is lengthened whereas a value of $\alpha < 1$ corresponds to a shortened signal.

2.1 Sinusoidal Model for Audio Signals

The underlying hypothesis as presented by McAulay et al. in [22], upon which current methods for time-scaling of a signal $x(n)$ rely, is that the signal $x(n)$, sampled at frequency F_s, is a sum of $P(n)$ sinusoids, called *partials*, with the number of partials $P(n)$ varying over time. It can be written as [23]:

$$x(n) = \sum_{p=0}^{P(n)-1} A_p(n) \cos(\frac{n}{F_s}\omega_p(n) + \phi_p) \tag{2.1}$$

Each partial has its own angular frequency $\omega_p(n) = 2\pi f_p(n)$, amplitude $A_p(n)$ and phase ϕ_p. The parameters are presumed to vary relatively slowly over time so that the signal is quasi-stationary over durations of a few milliseconds to a few dozens milliseconds, which corresponds well to what we know of speech and music.

According to this model, the goal of time-scaling is to change the rate at which $\omega_p(n)$ and $A_p(n)$ vary as a function of n, the time index. In other words, it is a resampling of each of these parameters by a factor equal to the speed ratio which can be constant α or time-varying $\alpha(n)$. Note that in the case it is time-varying, its variations are assumed to be relatively slow compared to those of the signal [20].

Harmonic Signals

In the following sections the term *harmonic*, applied to an audio signal, indicates that, at each instant n, the frequencies of all the partials $\omega_p(n)$ are

integer multiples of $\omega_0(n) = 2\pi f_0(n)$, with $f_0(n)$ the fundamental frequency. Examples of harmonic signals are single sound sources such as speech, singing, some mono-instrumental music with the notes being played one at a time, etc.

2.2 Time domain

Time-domain methods such as *synchronized overlap-add* (SOLA) [24], *waveform similarity-based synchronized overlap-add* (WSOLA) [25], *synchronized overlap-add, fixed synthesis* (SOLAFS) [26], *time-domain pitch-synchronous overlap-add* (TD-PSOLA) [27] and their variants are usually applied to harmonic signals, for instance speech and singing recordings. The basic principle of these methods in order to extend the duration of a signal is to segment the signal into overlapping frames and either duplicate some frames or increase the shift between each frame, as illustrated in Figure 2.1. Conversely when reducing the duration they either drop frames or reduce the frame shift. Most of the methods use frames with constant duration, except for TD-PSOLA and its variants which use frames centered on *pitch-marks* and whose duration is proportional to the fundamental period, hence the "pitch-synchronous" denomination.

2.2.1 Analog Domain

The principle of frame duplication and discard behind time-domain time-scaling was initially proposed in the analog domain more or less at the same time by several researchers such as Fairbanks et al. [28] or Gabor [29][2]. To put it simply, in a mechanical embodiment of Figure 2.1, they modified magnetic tape recorders so that it would drop or repeat chunks of sound while playing. As a result the output audio would have a different duration than the original.

[2] Although, in all fairness, Gabor's work is dedicated, among other things, to frequency-shifting and bandwidth usage in sound transmission, and does not explicitly mention time-scaling which is but a by-product of his method.

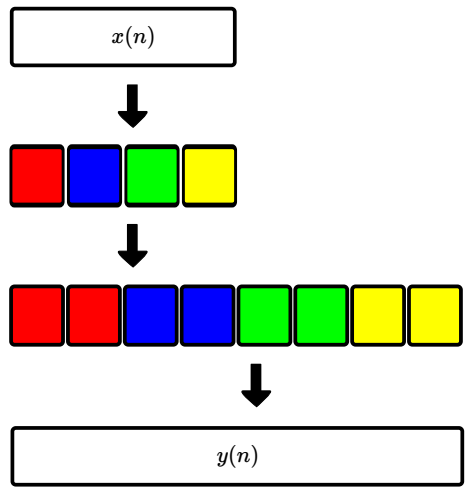

Figure 2.1. *In its simplest and oldest implementation, time-scaling consists in dividing an input signal $x(n)$ into frames and duplicating them to obtain an output signal, $y(n)$, with a different duration.*

2.2.2 Resampling

A simple example of time-scaling in the analog domain is that of the vinyl turntable when a disc is not played at the appropriate speed. However, this cause a distortion in the frequency. For instance a 33rpm record played at 45rpm (or any speed faster than 33rpm for that matter) has all its frequency content shifted toward higher frequencies.

In the digital domain this has an equivalent which is obtained by playing a signal at a different sampling frequency than the one it was recorded at. It causes a scaling of the frequencies, upward when accelerating (i.e. increasing the sampling frequency and thus shortening the duration of the playback) and downwards when decelerating (i.e. decreasing the sampling frequency and thus extending the duration of the playback). In practice though, for a digital signal $x(n)$, instead of modifying the sampling rate, this is achieved through a resampling of $x(n)$ by a factor equal to the speed factor α as shown in Figure 2.2. The resampled signal is then played at the original sampling rate [30].

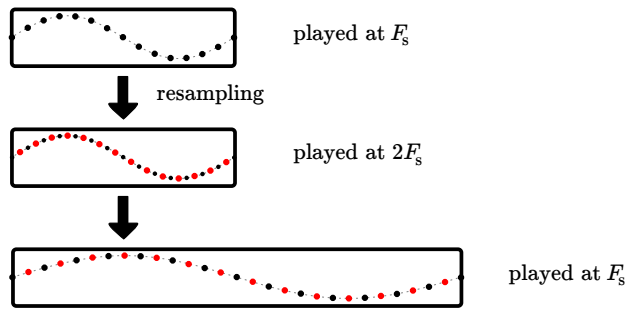

Figure 2.2. *A waveform (here a sine) is resampled by a factor α (here $\alpha = 2$). If the samples are played at twice the orignal sampling frequency, the sound is the same. If it is played at the original sampling frequency, it lasts twice as long. However, the frequency of the sine is divided by a factor two and the resulting sound is different from the original.*

This has been used frequently in movies as a special effect to arguably add either drama or a comical effect to slow motion and "fast-forward" scenes, hence with a shift of the frequencies. This is probably the reason why people not familiar with the domain, when asked about speeding up or slowing down a sound, mention the low-pitched voice of slow motion videos as the first example that pops in mind, most of the time. It is also the only audio signal currently available during a slow motion video in broadcast television, but it is rarely used because of the frequency shift.

2.2.3 SOLA

In general, a time-scaling method implemented as shown in Figure 2.1 would present discontinuities (*clicks*) at the junction between two frames. In the third part of his *Theory of Communication* [29] Gabor proposes that each frame fades in and out gradually, combined with a superposition (*overlap*) and summation (*addition*) of several frames. In the digital world this is implemented through a method called *OverLap-Add* (OLA), illustrated in Figure 2.3. Although this is an improvement over a simple concatenation of frames, in the sense that there are no discontinuities anymore in the time domain between two successive frames, the resulting audio can still present important acoustical discrepancies in the overlapping regions. Taking into account the

sinusoidal model from Section 2.1, this is due to phase discontinuities. Indeed, as shown in Figure 2.4, the sinusoidal components of two overlapping frames are not necessarily in phase and they may interfere destructively. For human ears this translates into an audible and localized amplitude modulation compared to the rest of the signal.

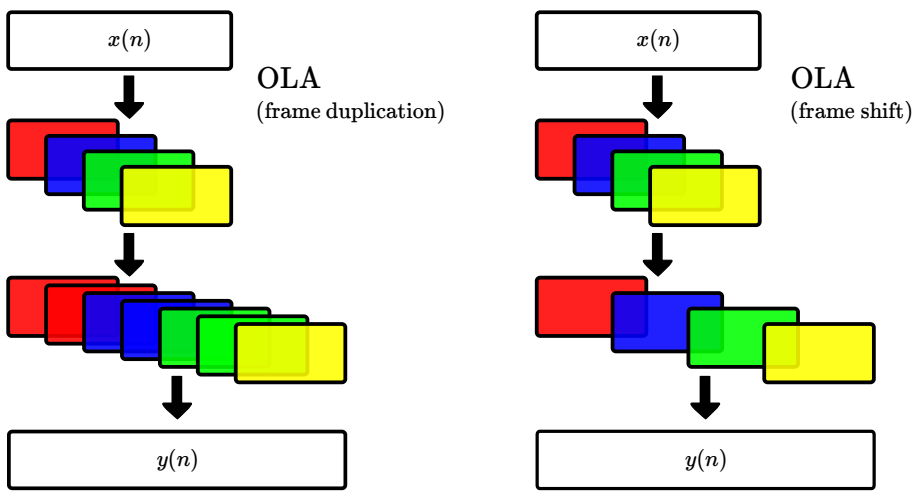

Figure 2.3. *Two different approaches to time-scaling using OverLap-Add synthesis. In one case extra frames are inserted (left) whereas in the other case existing frames are re-spaced (right) to match the stretching factor.*

In [24] Roucos and Wilgus develop the *Synchronized OverLap-Add* (SOLA) of time signals, schematized in the left graph of Figure 2.5. This algorithm adds an extra step to OLA: instead of being overlap-added at an arbitrarily fixed place, as it is in OLA, each frame is inserted at a time position located around that fixed place so that it maximizes its correlation to the signal that has already been generated, hence reducing the phase incoherences. In order to do so, it uses the maximum of a cross-correlation measure between the frame to be inserted in the output signal and the end of said output signal (in practice the samples of the frame that was inserted just before). The position of this maximum in the cross-correlation determines the optimal position for the frame to be inserted.

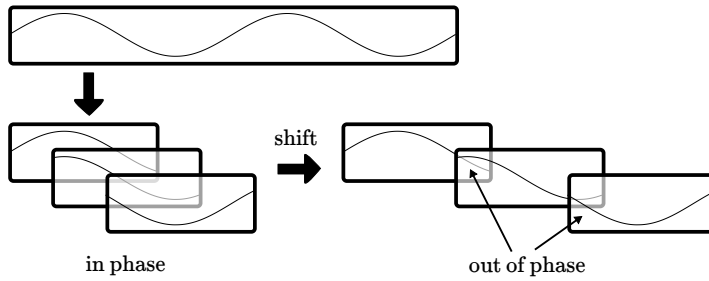

Figure 2.4. *An input sinusoid is divided into overlapping frames. The chunks of sinusoid in each frame are in phase with each other and if the frames are overlap-added, the original sine is obtained. However, if the shift between frames changes, the sines are now out-of-phase.*

In the case of harmonic signals such as voiced regions of speech, this is equivalent to a resynchronization of the frames with the pitch of the signal [31]. It also explains why time-domain algorithms are usually used only with harmonic signals: with polyphonic signals only a part of the underlying harmonic signals would be resynchronized by the cross-correlation. Other parts of the signal would still produce a modulation that, if audible, reduces the quality of the time-scaled signal.

2.2.4 SOLAFS and WSOLA

More or less simultaneously Hejna et al. [26, 32] and Verhelst et al. [25, 33] developed an approach to time-scaling that is somewhat complementary to SOLA. Hejna's method is named *Synchronized OverLap-Add, Fixed Synthesis* (SOLAFS) and Verhelst's is described as a *Waveform Similarity-based OverLap-Add* (WSOLA) method. In practice the two are all but identical[3] [34] and their principle is illustrated on the right graph of Figure 2.5.

During the synthesis, this algorithm overlap-adds frames at regular intervals, like OLA, but these frames are chosen in the input signal in the neighborhood of the frame that would have been used by OLA, using a cross-correlation

[3] WSOLA computes a correlation measure using only frames from the input signal whereas SOLAFS computes the correlation measure between samples of the input signal and samples from the overlap region of the output signal.

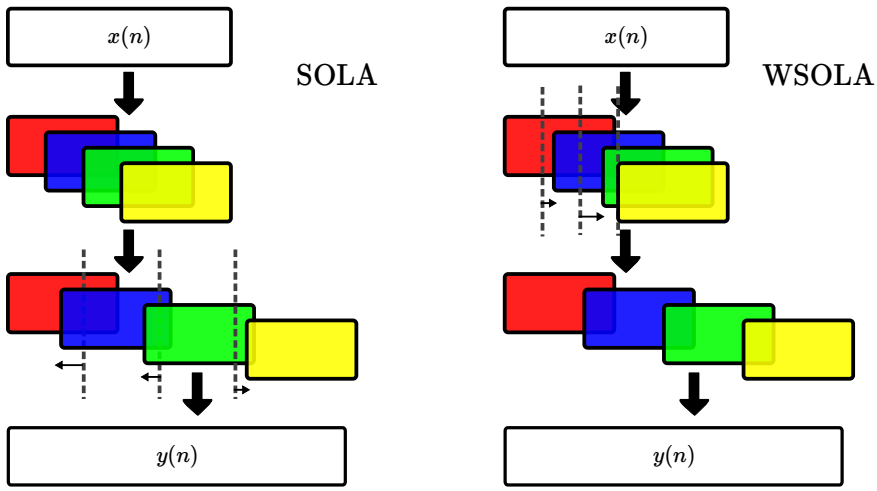

Figure 2.5. *In the SOLA method, each synthesis frame can be slightly translated in time around its theoretical insertion position (dashed lines) to minimize discontinuities. In the SOLAFS/WSOLA method, it's the opposite, each analysis frame can be selected slightly around its theoretical position (dashed lines), but the synthesis hopsize is constant.*

measure just like SOLA. This contrasts with SOLA which divides the input signal into regularly-spaced frames that are then reassembled at an irregular pace to synthesize the output signal. In other words, SOLA uses constant analysis frame shifts and variable synthesis frame shifts whereas SOLAFS and WSOLA do the opposite.

2.2.5 TD-PSOLA

The *Time-Domain Pitch-Synchronous OverLap-Add* (TD-PSOLA) algorithm is described by Moulines and Charpentier [27, 31, 35] in three steps.

The first step, the *pitch-synchronous analysis*, decomposes the signal into overlapping frames centered on so-called *pitch marks*. There is one pitch mark per period of the speech signal and thus the spacing between two successive pitch marks is equal to the local pitch period. Each frame has a length between 2 to 4 pitch periods and is windowed, often by a Hann window [36].

The second step, the *pitch-synchronous modification*, consist in creating a new sequence of pitch marks that preserves the pitch spacing of the original pitch marks, but extends or reduces the signal duration. In other words, pitch marks are duplicated or dropped as a function of the time-scaling factor to create the new sequence of pitch marks while preserving the pitch evolution over time [36]. Each new pitch mark is thus linked to a pitch-synchronous frame.

Finally, the third step, the *pitch-synchronous overlap-add synthesis*, overlap-adds the new sequence of pitch-synchronous frames in order to synthesize the time-scaled signal. Since each frame is pitch-synchronous the parts that overlap-add are, theoretically, perfectly in phase with each other and no destructive interferences can happen, hence leading to a high quality modification of the time scale for factor $0.5 < r < 2$. Larger factors are possible for the slow down but require specific processing of the unvoiced regions.

Although previous algorithms are based on an underlying sinusoidal model, they can still be applied to any kind of sound. However, the TD-PSOLA algorithm is specifically targeted at signals presenting a pitch, such as speech and singing. It is therefore not applicable to sports recordings such as those presented in Chapter 4 which are mostly made up of noise.

2.3 Frequency domain: the phase vocoder

The basic idea of the method in the frequency domain is the same as for the time-domain OLA: duplicating some frames or increasing the frame shift to decelerate the recording, dropping some frames or reducing the frame shift to accelerate the recording. However, instead of adjusting the position of the frames in the time domain to reduce phase discontinuities, as it is done in SOLA-like methods, the frequency-domain approach directly modifies the phases of each frame in the spectral domain so that the OLA happens seamlessly. This method is called the *phase vocoder* and its implementation is generally achieved through the short-time Fourier transform and its inverse, which are defined in Section 1.2.3.

2.3.1 Phase vocoder

The *phase vocoder* is presented in 1966 by Flanagan and Golden [37] as an evolution of the *channel vocoder* invented by Dudley [38, 39]. The approach proposed by Flanagan et al. is the so-called "filter-bank" approach, where the input signal $x(n)$[4] is passed through a bank of band-pass filters, each filter covering a different frequency band so that the parameters, amplitude and phase[5], representing each frequency band of the signal as a function of time can be computed. In a sense, the parameters are similar to those of the parametric representation in Equation 2.1 and an approximation of the original signal can be resynthesized through additive synthesis using an oscillator bank.

In order to obtain a time-scaling effect, the frequencies and phases of the oscillator bank are multiplied by a speed factor α and the signal is resynthesized. Then the signal is played $1/\alpha$ times faster. As explained in Section 2.2.2, this multiplies the frequencies by a factor $1/\alpha$ thus compensating for the previous multiplication by α. Consequently the signal has the same spectral content as the original one. However, since it is played at $1/\alpha$ times the original speed it has α times the original duration. In practice this is an inefficient implementation and, in 1973, Schafer and Rabiner [40] make use of the STFT to implement the filter-bank[6] analysis, followed by Portnoff [41], in 1976, who improves over the method by using the inverse STFT for the synthesis as well.

Different implementations of the STFT-based vocoder exist and are described in the following, but in all cases the audio signal $x(n)$ is divided into overlapping frames $f_i(n)$ which are multiplied by a window function $w(n)$. The spectrum $F_i(k)$ of each frame is computed to fill the *analysis* STFT matrix $S_a(i,k)$ and then, depending on the implementation, either the phases alone or the phases, amplitudes and number of frames of $S_a(i,k)$ are modified to obtain $S_s(i,k)$, the *synthesis* STFT, on which an inverse STFT is applied to resynthesize a time-scaled audio signal. During the inverse STFT, each synthesis frame computed is again multiplied by a window, generally the same window $w(n)$ that is used during the analysis. Note that in order to simplify

[4] Note that in [37] the reasoning is applied first to an analog signal $x(t)$ before being extended to a digital one.

[5] Whereas Dudley's vocoder would only retain the amplitude, hence the name *phase* vocoder.

[6] Hence benefiting from the speedup provided by the FFT.

the notation and because each frequency bin k is treated the same way in a standard phase vocoder, the notations $S_a(i)$ and $S_s(i)$ represent respectively $S_a(i,k)$ and $S_s(i,k)$ in the following.

Frame shifting

The most common implementation of the phase vocoder found in the literature [2, 23, 42, 43] uses different sizes for the shift between frames (*hopsize*) during the analysis and the synthesis steps. As is illustrated in Figure 2.6, the ratio between these two frame shifts equals the desired slow-down/speed-up factor. This means that in order to change the speed by a factor α, given a synthesis hopsize R_s and an analysis hopsize R_a, they must verify Equation 2.2.

$$\alpha = \frac{R_s}{R_a} \tag{2.2}$$

As explained in Section 2.2.3, since the relative position of each frame in the output signal is different from that of the frames in the input signal, a simple overlap-add of the frames to generate that output causes phase discontinuities and thus destructive interferences. The main idea behind the STFT implementation of the phase vocoder is to adapt the phase of each partial of Equation 2.1 according to the new hopsize R_s so that all the frames overlap seamlessly. Roughly speaking the adaptation needs to keep the variation of phase constant over time.

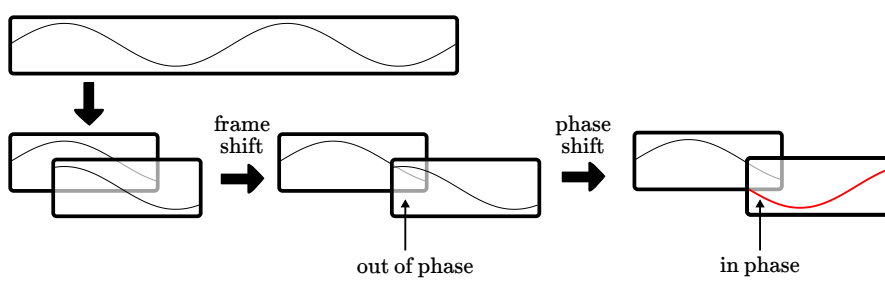

Figure 2.6. *When the hopsize changes, the frames become out of phase. The vocoding modifies the phase of each frequency bin so that they are in phase again.*

For each bin k of the STFT, the phase variation between input frames i and $i-1$ is compared to the expected phase variation for that bin (a function of k and R_a). The difference between these two values (the *heterodyne phase increment*) is converted to the range $\pm\pi$ (Equation 2.6), divided by R_a and added to the theoretical phase variation for bin k in the output signal (a function of k and R_s). Finally, this value is added to the phase of output frame $i-1$ to obtain the phase of output frame i (Equation 2.7). Note that the input frame 0 is reused as output frame 0 (Equation 2.3) and that the spectral amplitudes are not modified (Equation 2.4).

$$S_s(0) = S_a(0) \tag{2.3}$$

$$|S_s(i)| = |S_a(i)| \tag{2.4}$$

$$\Omega = \{0, \ldots, k\frac{2\pi}{L}, \ldots, (L-1)\frac{2\pi}{L}\} \tag{2.5}$$

$$\Delta\phi(i) = [\angle S_a(i) - \angle S_a(i-1) - R_a\Omega]_{2\pi} \tag{2.6}$$

$$\angle S_s(i) = \angle S_s(i-1) + R_s(\Omega + \frac{\Delta\phi(i)}{R_a}) \tag{2.7}$$

where $S_a(i)$, $S_s(i)$, Ω and $\Delta\phi(i)$ are L-sample vectors with L the length of a frame. $[\,]_{2\pi}$ denotes the conversion of the phase to the range $\pm\pi$ [23].

Once the DFT of a synthesis frame has been calculated, it is overlap-added to the output signal by an inverse STFT, using $w(n)$ as the synthesis window.

Frame generation

Another implementation of the phase vocoder is proposed by Bonada in [44] and Ellis in [45]. As Figure 2.7 shows, contrary to the previous method it uses the same shift between the frames at analysis and synthesis time. Obviously when doing time-stretching the number of frames used to synthesize the output is different from the number of frames extracted from the input. Frames have to be dropped or created one way or another. In the algorithm as implemented by Ellis, all frames are generated by interpolating the spectral amplitudes and accumulating the phase variations between the analysis frames.

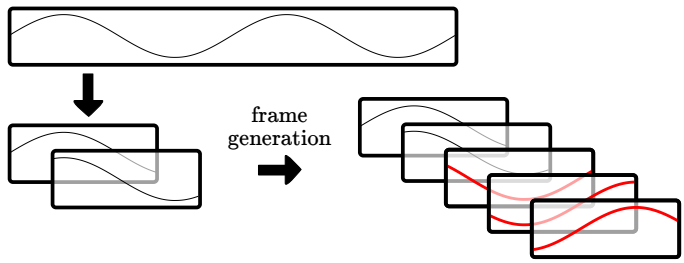

Figure 2.7. *In this case, the hopsize is constant and extra frames are generated (in red) using the phase information contained in the two analysis frames.*

The first step sets the initial synthesis frame spectrum $S_s(0)$ equal to the initial analysis frame spectrum $S_a(0)$.

$$|S_s(0)| = |S_a(0)| \tag{2.8}$$
$$\angle S_s(0) = \angle S_a(0) \tag{2.9}$$

For the following synthesis frames the analysis and synthesis indices, i and j, are updated according to Equations 2.10 and 2.11, respectively.

$$i = i + \frac{1}{\alpha} \tag{2.10}$$
$$j = j + 1 \tag{2.11}$$

i is generally not an integer value. For instance if the speed factor α is 2 (2× slower), $S_s(7)$ corresponds to a frame position, in the original audio, equal to $7/\alpha = 3.5$. In other words, a "virtual" frame located between $S_a(3)$ and $S_a(4)$.

The spectrum $S_s(j)$ of the j^{th} synthesis frame is a function of the amplitude and phase variations of its "surrounding" analysis frames as well as $\angle S_s(j-1)$:

$$\lambda = i - \lfloor i \rfloor \tag{2.12}$$
$$|S_s(j)| = (1-\lambda)|S_a(\lfloor i \rfloor)| + \lambda |S_a(\lfloor i \rfloor + 1)| \tag{2.13}$$
$$\Delta\phi(i) = [\angle S_a(\lfloor i \rfloor + 1) - \angle S_a(\lfloor i \rfloor)]_{2\pi} \tag{2.14}$$
$$\angle S_s(j) = \angle S_s(j-1) + \Delta\phi(i) \tag{2.15}$$

where $\lfloor i \rfloor$ is the integer value of i. Finally, the inverse STFT over $S_s(j,k)$ overlap-adds every frame into the output signal.

This second implementation has some interesting advantages. For $\alpha > 1$, the matrix $S_a(i,k)$ is smaller than the one computed in the first implementation, thus requiring less DFT computation.[7] Besides, it can interpolate as many synthesis frames as needed at a given time position i, which means that it can stay at that playback position for an infinite duration, literally pausing in the signal while still playing it. The first proposed implementation based on frame shifts does not make this possible. Also, note that if the speed factor α is constant, the analysis indices i can be linearly mapped to the synthesis indices j using Equation 2.16.

$$i = \frac{j}{\alpha} \tag{2.16}$$

2.3.2 Phase-locked vocoder

Compared to time-domain approaches, the phase vocoder has the advantage to work with both mono and polyphonic signals. Besides, it theoretically overlaps frames perfectly in phase with each other. In practice, however, it produces a sound that can be perceived as muffled, reverberant and/or moving away from the microphone [43, 46], especially for time-stretching. This distortion is called *phasiness* [47] and the accepted explanation for its presence is a loss of coherence between the phases across the bins of the short-time Fourier transform over time, also called *loss of vertical phase coherence*.

Indeed, the methods presented in Section 2.3.1 are applied independently to each bin k of the spectrum in order to keep intact the phase constraints along the time (or horizontal) axis of the spectrogram. As a consequence there is no constraint with regard to the frequency (or vertical) axis: if there is a dependency between neighboring bins $[\ldots, k-1, k, k+1, \ldots]$, in the input signal it is lost in the process. This causes the phasiness artifact [47].

Several algorithms have been proposed in order to correct this problem. For instance, Puckette [46] uses the phase of the sum of the spectral values from

[7] More exactly, the size of $S_a(i,k)$ is constant in this case, it is the matrix $S_a(i,k)$ used for the frame shifting vocoder that grows in size with α.

bins $k-1$, k, and $k+1$ as the final phase value $\angle S_s^*(i)$ for bin k:

$$\angle S_s^*(i,k) = \angle(S_s(i,k-1) + S_s(i,k) + S_s(i,k+1)) \tag{2.17}$$

Laroche and Dolson [43] proposed a somewhat more complex approach called *identity phase locking*: the peaks in the spectrum are detected and the phases $\angle S_s(i,k_p^i)$ of their corresponding maximum bins k_p^i are updated normally by the phase vocoder. The other bins located in the region of influence of k_p^i have their phases modified so as to keep constant their phase deviation from the phase of k_p^i. The phase propagation equations for these non-peak bins are

$$\angle S_s(i,k) = \angle S_s(i,k_p^i) + \angle S_a(i,k) - \angle S_a(i,k_p^i) \tag{2.18}$$

$$\angle S_s(j,k) = \angle S_s(j,k_p^i) + \angle S_a(i,k) - \angle S_a(i,k_p^i) \tag{2.19}$$

respectively for the "frame shifting" (Equation 2.18) and the "frame generation" (Equation 2.19), with i and j the indices as defined in Equations 2.10 and 2.11 for the second case. As a result of these constraints, there is a horizontal phase locking for the peaks and a vertical phase locking for all the other bins of the spectrum.

A refinement of this method, also presented by Laroche and Dolson, is to track the trajectories of the peaks over time and use the previous phase of each maximum bin k_p^{i-1} to compute that of the next one k_p^i. Equations 2.6 and 2.7 become

$$\Delta\phi(i) = [\angle S_a(i,k_p^i) - \angle S_a(i-1,k_p^{i-1}) - R_a\Omega]_{2\pi} \tag{2.20}$$

$$\angle S_s(i,k_p^i) = \angle S_s(i-1,k_p^{i-1}) + R_s(\Omega + \frac{\Delta\phi(i)}{R_a}) \tag{2.21}$$

where k_p^i is the bin index of the p^{th} peak in analysis frame i. This is important if the bin indexes k_p^{i-1} and k_p^i for peak p are different between analysis frames $i-1$ and i as it prevents the phase of the peak at frame i from being based on the phase of a non-peak bin at frame $i-1$. However, tracking peaks over time is not always straightforward (peaks can appear, disappear, split or merge which increases the complexity of the task).

For this Laroche et al. propose to consider that the peak k_p^{i-1} is the peak to which bin k_p^i belonged to at time $i-1$. They also propose to update the

phases of the neighboring bins as

$$\angle S_s(i,k) = \angle S_s(i,k_p^i) + \beta[\angle S_a(i,k) - \angle S_a(i,k_p^i)] \tag{2.22}$$

where β is a scaling factor, giving the method the name *scaled phase locking*.

Note that in the implementation where the frames are generated, the integration of scaled phase-locking is a bit more tricky since the analysis frames $S_a(\lfloor i \rfloor + 1)$ and $S_a(\lfloor i \rfloor)$ can be the same for several synthesis frames $S_s(j)$ in a row, in which case no peak-tracking phase locking should happen, except for each first new value of $\lfloor i \rfloor$. In other word the value $\lfloor i \rfloor$ can remain the same for successive values of j, and peak tracking should happen only whenever a new value of j brings a new value of $\lfloor i \rfloor$.

In [48], Karreer et al. refine the phase-locking with two improvements: *multiresolution peak-picking* and *sinusoidal trajectory heuristics*. On the one hand, the peaks are detected with a multiresolution method, in other words, they are picked within smaller regions in the lower frequencies of the spectrum than in the high-frequency parts. As a result more peaks will be detected in the low-frequency parts than in the high-frequency ones. On the other hand, they define a set of additional rules for the tracking of spectral peaks across frames in order to reduce the false detection of peak trajectories. Finally, they add a third improvement, called *silent passage phase reset* which consists in setting all the phases of a synthesis frame to those of the corresponding analysis frame during silent part of the signal, hence cancelling any vertical phase incoherences. Such an abrupt reset of the phase should cause an audible *click*, but since it happens in silent region of the signal it is not perceived by the listeners.

For small lengthening ratios, Dorran et al. [49] recover phase coherence by slightly adjusting the phase of each synthesis frames so that after a few frames it converges to an almost perfect overlap with one of the analysis frames. From that point on, a group of frames from the original signal can be added directly to the output signal without any phase transformation. This results in a (locally) perfect-quality audio signal. The gradual adjustment of the phases is calculated in order to be imperceptible to human ear.

Other methods use the phase vocoder but mix it with approaches from different domain. Those are introduced in Section 2.5. Among these, following upon

the idea of phase reset, we published in 2011 a novel approach to the problem of phasiness [50] and, in 2012, Kraft et al. proposed an improved version of the method [51]. We named the method PVSOLA and it is detailed in Chapter 3.

2.4 Model-based

Time-domain methods rely on the parametric model of Equation 2.1 but do not directly use it in their inner working. Even TD-PSOLA only needs to know the local pitch, which for normal speech is usually equal to $\omega_0(n)$, but none of the other parameters. The phase vocoder, on the other hand, processes every bin k of the STFT as a sinusoidal component without discrimination, although the phase lockings proposed by Laroche et al. make a distinction between significant peaks and other bins.

In between these two extremes lies the model-based approach that analyzes the input signal to extract the features of a parametric model, Equation 2.1 being just one example of such a model. From the model and its parameters a signal can then be synthesized with as many alterations to the parameters as needed, creating various effects such as time-scaling, for instance. Note that in order to use a model-based approach one needs an a priori knowledge of the signal to model. Modeling the percussion of a drum as if it were a flute would make little sense. In the following sections we introduce some of the most common models for analysis and resynthesis of sounds that can be used for time-scaling.

2.4.1 Source-Filter Model

The source-filter model [52] represents a sound as the result of an excitation signal, the *source*, passed through a *filter* that modifies its spectral envelope. It is a widespread representation of voice signals in speech processing where the source represents the action of the vocal folds and the filter that of the vocal tract.

The source is usually modeled as either a sequence of harmonic signals (pulse trains, glottal pulses, etc.) for voiced parts of speech or a white noise for unvoiced parts, or a weighted mix of both (mixed excitation, CELP codebooks,

etc.). In source-filter models, filtering often implies time-domain filtering, but it is also possible to apply filtering in the spectral domain. For instance, an STFT of the source, followed by a modification of the spectrum according to the spectral envelope of the filter and, finally, an inverse STFT synthesizes a filtered signal.

In the time-domain type of filtering, the filter parameters are usually obtained through a *linear prediction analysis* (or a variant of it) of each frame from the original signal, whereas for spectral filtering, the envelope of the filter can be obtained by an STFT analysis of the original signal, followed by either a smoothing of its spectral amplitude, a measure of band-by-band spectral energies, a peak-to-peak interpolation of the spectrum, etc.

As an example, the most common parameters used in speech analysis and synthesis are the voiced/unvoiced decision, the pitch, the energy and the coefficients of the linear filter. Afterwards, these parameters can be resampled by a factor equal to the speed factor α so that a resynthesis using these resampled parameters last α times longer.

Note though that linear interpolation is often preferred to resampling for the pitch and energy since these parameters are assumed to vary slowly, at least in speech, compared to the rate at which the frames are extracted from $x(n)$. As for the linear prediction filter, a resampling or an interpolation of its coefficients may result in unstable filters, therefore safer albeit convoluted interpolation methods are generally applied. The coefficients are transformed into others, more stable (parcor, cepstrum, line spectral pairs, etc.) which can be interpolated more safely. Then these are transformed back into linear prediction coefficients. In other words, the term "resampling" ought to be considered in its loosest acceptation (a change in the number of "samples" of each parameter) as opposed to its strict Nyquist-Shannon-compliant definition.

2.4.2 Sinusoidal Model

The simplest form of a sinusoidal model is already presented in Section 2.1. Most time-domain algorithms and the phase vocoder rely on it. In a model-based time-scaling framework each of the parameters $P(n)$, $\omega_p(n)$, $A_p(n)$ and ϕ_p of Equation 2.1 is extracted from the signal, generally by spectral analysis. From these parameters, it is possible to resynthesize an approximation

of the original signal[8] or a time-scaled version of it, among other potential transformations.

As an extremely simplistic example, a pure tone can be modeled with a single sinusoid ($P(n) = 1\ \forall n$), with a given frequency ω and amplitude A that can be deduced, for instance, from a Fourier analysis of the tone. Using these two parameters, one can generate a sinusoid that approximates the original signal for as long as needed. Time-scaling a one-second pure tone is then just a matter of synthesizing two seconds of the parametric sinusoid, changing the pitch of the tone is a change of ω away, and so on.

Generalization to any constant value of P follows immediatly as each sinusoid is processed independently. When the number of sinusoids vary over time or when their frequencies change enough that they cross trajectory in the spectral domain, advanced tracking heuristics must be put in place in the analysis part [22, 53], similarly to what is needed for the phase-locking described in Section 2.3.2.

Sine + Noise Model

In 1989, Serra published his PhD thesis [53] in which he presents a *deterministic plus residual model* (DR) and a *deterministic plus stochastic model* (DS), later named *spectral modeling synthesis* (SMS), that take into account the noisy components present in an audio signal $x(n)$ as an additive component $e(n)$ to the sinusoidal model from Equation 2.1

$$x(n) = \sum_{p=0}^{P(n)-1} A_p(n) \cos(\frac{n}{F_s}\omega_p(n) + \phi_p) + e(n) \tag{2.23}$$

where $e(n)$ is named the residual or the stochastic part depending on the method. In the first case, it is assumed to contain anything that is not a sine, which may include non-stochastic events such as transients[9], whereas in the

[8] Synthesis of a sound by applying Equation 2.1 "as is" is a form of *additive synthesis*, as opposed to the filtering of a source as described in Section 2.4.1 which is *subtractive synthesis*.

[9] Note attacks, plosives, etc.

second case it is assumed to be completely stochastic. Also, note that for the DS model, the phases of the partials are discarded.

In the case of the DR model, the spectral envelope of $e(n)$ can be obtained by first estimating the parameters of the sinusoidal components and then subtracting the sum of these components from $x(n)$. However, in the case of the DS model, since the phases of these components are dropped, it is not possible to make the subtraction in the time domain. Therefore, the amplitude spectrum of the sum of partials (i.e. $A(k)$) is subtracted from $X(k)$, the amplitude spectrum of $x(n)$, to obtain directly $E(k)$, the spectral envelope of the stochastic part $e(n)$. From this spectral envelope it is possible to resynthesize the residual/stochastic part using either a filtered white noise or an inverse STFT of $E(k)$ with random phases.

Time-scaling of the harmonic content is performed as explained in Section 2.4.2 whereas the noisy part is time-scaled by "resampling"[10] the parameters representing the spectral envelope of $e(n)$ (whether it is the spectral envelope $E(k)$ itself or a time-domain filter) which can then be used to synthesize a time-scaled version of $e(n)$, as explained in Section 2.4.1.

Sine + Transient + Noise Model

The SMS model does not account for transient events such as a note attack in a musical instrument, a plosive consonant in speech, a drum beat, etc. Verma et al. proposed the *transient modeling synthesis* (TMS). TMS is an improved version of SMS that models transients hence allowing time-scaling of such signals [54].

The algorithm adds an intermediary step between the computation of parameters of the partials and the extraction of the spectral envelope of the stochastic component. Using a DCT analysis over the residual signal obtained by the subtraction of the sinusoidal components from $x(n)$, it identifies transients. Indeed, according to Verma et al., a transient in the time domain is transformed into a sinusoidal curve in the DCT domain (frequency). This sine can therefore be modelized with the same sinusoidal model that is used for the partials. These synthetic transients are then removed from the residual to obtain the stochastic component of the signal, that is the component that

[10] For some loose definition of resampling, see Section 2.4.1.

contains neither partials nor transients. As for the resynthesis step, the sinusoids corresponding to each transient are generated using the sinusoidal model an passed back as transients into the time domain by an inverse DCT.

Time-scaling of the partials and the stochastic part is performed as explained in the previous sections. For transients, the time-scaling happens on the synthetic sinusoids in the DCT domain, which are eventually transformed into the time domain, thus becoming transients which are added to the signal made of the time-scaled partials and stochastic part. The time-scaling of the sines in the DCT domain ensures that each transient is translated to its proper location in the time domain given a speed factor α.

Harmonic plus Noise Model

Introduced by Stylianou [55,56] in 1995, the *harmonic plus noise model* (HNM) also represents a speech signal $x(n)$ as the sum of a deterministic and a stochastic part: up to a certain frequency $F_{max}(n)$, the *maximum voiced frequency*, it models the spectrum of the signal with sinusoidal harmonics while above that frequency the spectrum is modeled by a noise excitation passed through a cascade of time-varying filters. The cascade is composed of a linear prediction filter modeling $x(n)$ and a high-pass filter whose cutoff frequency is $F_{max}(n)$.

The system is represented by the same Equation 2.23 as Serra uses, but the constraints are that $\omega_p(n) < F_{max}(n)$ and $e(n)$ is a filtered white noise whose frequency content is negligible below $F_{max}(n)$, hence the high-pass filtering, and matches the spectral envelope of $x(n)$ above $F_{max}(n)$. Time-scaling the harmonic content is performed as explained in Section 2.4.2 whereas the noisy part is time-scaled by "resampling" the filter parameters as in Section 2.4.1.

2.4.3 Physical Model

Physical models can be either an actual physical reproduction of a system producing a sound, such as a model of our vocal apparatus that mimics speech, or a software implementing mathematical models of physical phenomenon (using equations of sound propagation and reverberation in fluids, among others) to synthesize sounds that ressemble the real ones. It does not seem realistic to us

to attempt to model a complete football stadium and its occupants, much less in realtime. Therefore, the physical models are not addressed in this work.

2.5 Mixed methods

Some methods use a combination of ideas from several of the three different domains. For instance, Dorran et al. [49], already introduced in Section 2.3.2, use a modified phase vocoder but once in a while insert frames from the original signal directly in the time domain. In 2010, Röbel presents an enhanced version of SOLA [57] where a phase vocoder is used to modify the phases of each frame so that they overlap properly instead of adapting their position in the output audio signal thanks to a modified cross-correlation measure.

In 2011 we combined a phase vocoder with a SOLA method [50], in a way that we see as orthogonal to Röbel's approach, to regularly insert frames from the original signal into the output signal, significantly reducing the phasiness artifact for harmonic signals. Kraft et al. improved our method in [51] to speed it up and generalize it to polyphonic signals.

Most arguably, PSOLA-based methods and the identity or scaled phase locking algorithms might be seen as mixed models. Although PSOLA itself is, strictly speaking, a time-domain only algorithm, it needs pitch marks which are often obtained through frequency analysis. Also phase locking is halfway between a phase vocoder and a sinusoidal model, although no sinusoidal additive synthesis happens.

Finally, note that the new method to time-scaling of sports event recordings that we developed during our research and which is presented in Chapter 6 combines time and frequency-domain or time and model-based approaches.

2.6 Transient Detection and Time-Scaling

2.6.1 Definition

A transient signal is a perceptual and contextual concept, and as such a subjective one, which makes it difficult to have a precise definition. In addition

to the tentative explanation hereafter, it is probably more appropriate to illustrate it through simple examples and to present it from the point of view of the time-scaling artifacts that we want to avoid.

From the signal processing perspective a transient signal could be defined as any signal that is not stationary or pseudo-stationary, for instance a transition region between two otherwise stationary signals[11]. However, and particularly in the field of audio processing, a transient also denotes a sudden, sharp, rapid[12] and perceptible variation of the content of the signal. Therefore, besides the non-stationary aspect, there is a speed aspect as well as a perceptual aspect that both have to be taken into account.

Considering this "definition" it becomes obvious, from the point of view of time-scale modifications, that a transient signal should not be time-scaled as any other signal. Indeed, the purpose of time-scaling is to change the playback speed of the signal without changing its spectral content whereas the way a transient is perceived is deeply linked to both its duration (or playback speed) and its frequency content. Therefore, transients are the parts of a signal that must not be time-scaled but instead "translated" to a new position in time, as opposed to quasi-stationary sinusoids and noise components.

In other words, time-scaling of transients does not have to consider the mathematical accuracy of the transformation, it just has to "sound right" to human listeners [58]. It is usually decomposed into three steps explained in Sections 2.6.2 to 2.6.4: a step computing a measure of *transientness* of the input signal, a step using this measure to detect the transients and their position and a step that extract and processes them separately from the rest of the signal.

Examples

In speech, typical examples for transients sounds are *stop consonnants* such as [p], [b], [t], [d], [k] and [g], glottal stops, etc. In music, signals to be preserved are rapid transitions between notes, drum-like sounds, note onsets such as the first milliseconds a piano note is hammered or a string of a guitar is plucked as shown on Figure 2.8, etc.

[11] Or a transition from a region with a zero signal to one with a non-zero signal.
[12] Typically between a few milliseconds and a few dozens milliseconds.

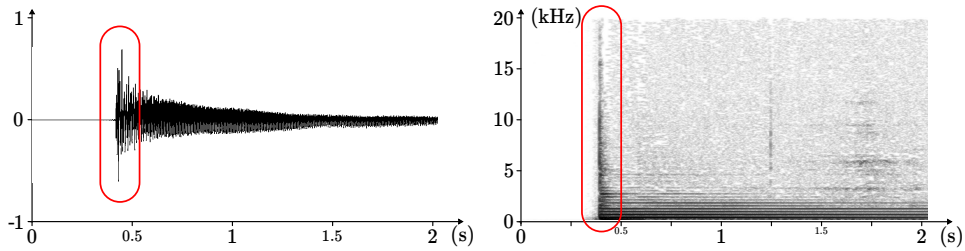

Figure 2.8. *Waveform (left) and spectrogram (right) of an attack (red boxes) for a single guitar string (A3).*

In sports recordings, impact sounds such as a kick in a ball, whistle or voice onsets, footsteps, applauses, etc. have to sound similar to the original ones in the time-scaled version. Besides, they have to be kept synchronous with the image frames of the video. We address these issues in Chapter 5 and 6.

Artifacts

When processed normally with the methods described in Sections 2.2 to 2.5, a transient can be subject to various artifacts illustrated in Figure 2.9, namely duplication, disapperance, desynchronization and smearing. For instance, a time-domain algorithm that repeats a frame to extend a signal also duplicates any transient present in the frame. Similarly, a frame dropped may result in the loss of the transients it contains, although in this case the transients are probably present in neighboring frames if there is an overlap between successive frames and might hence be preserved.

Algorithms such as SOLA and WSOLA may limit these problems for small changes of duration, up to a certain extent, because the cross-correlation-based resynchronization may happen to resynchronize a transient with itself across several successive frames, but the larger the stretching factor the more likely things are to go awry. On the other hand, these very same algorithms shift frames around their theoretical position to reduce phase mismatches. Hence transients can be shifted to slightly different positions than the one they should have been placed at. If these transients set a tempo in a musical piece, for instance drum beats, these slight variations may become annoyingly audible.

Another form of disapperance can occur in model-based time-scaling. Indeed, if the model does not take them into account the synthetic time-scaled signal most often lacks any transient. However, it might be modeled "accidentally", for instance if the variations of the parameters of the partials or the noise are sampled at a fast enough rythm to approximately reproduce the transients.

Finally, *transient smearing* is a term used to denote a spread of the transient energy over time and therefore a smoothing of its percussiveness, as opposed to an instant and sharp attack. It is a typical artifact of time-stretching with the phase vocoder and an example is shown in Figure 5.1.

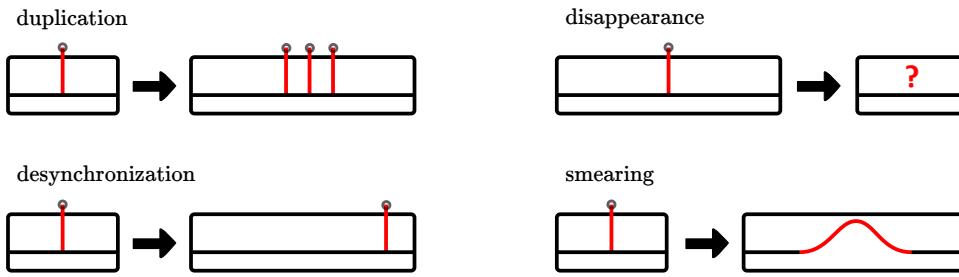

Figure 2.9. *Schematic representation of four artifacts for transients during time-scaling: duplication, disappearance, desynchronization and smearing.*

2.6.2 Detection

Most authors perform transient detection based on the selection of significant peaks in a *detection function* which is representative of the *transientness* of the signal over time. The function is obtained through various processings of the audio signal and each peak of the function represents the position of a possible transient. Whether a peak is significant depends mostly on the interpretation of the definition of a transient.

Many different functions have been published [59], generally in the fields of speech and music processing. It is not the intent of this section to detail them all. They go from finding local maximums of the waveform above a constant threshold to multi-band complex spectral analysis with adaptive thresholding

along with functions using perceptual models. Only the ones that are tested in Chapter 5, for transient detection in sports events, are presented here.

Energy

The local energy over time $E(n)$ as defined in Equation 1.6, its time derivative $D(n) = E(n) - E(n-1)$ or its normalized form $D(n) = 1 - \frac{E(n-1)}{E(n)}$ can be used as detection functions. Peaks in $D(n)$ correspond to sudden variation in the energy $E(n)$ with more abrupt variations corresponding to the higher peaks. As we will show in Chapter 5 this function performs poorly in noisy environment such as footbal stadiums, for instance.

Spectral Flux

The spectral flux is a measure of the positive variations of the spectral amplitudes. It is computed directly from the short-time Fourier transform (STFT) explained in Equation 1.15 of Section 1.2.3. Depending on the sources, one can find different although closely related definitions, some of which are presented in Equations 2.24 to 2.27

$$SF(n) = \sum_{k=0}^{N-1} H(|S(n,k)| - |S(n-1,k)|) \tag{2.24}$$

$$SF(n) = \frac{\sum_{k=0}^{N-1} H(|S(n,k)| - |S(n-1,k)|)}{E(n)} \tag{2.25}$$

$$SF(n) = \sum_{k=0}^{N-1} H(|S(n,k)| - |S(n-1,k)|)^2 \tag{2.26}$$

$$SF(n) = \sum_{k=0}^{N-1} H(|S(n+1,k)| - |S(n-1,k)|) \tag{2.27}$$

where $S(n,k)$ is the STFT defined in Equation 1.15 and $H(x) = \frac{x+|x|}{2}$, which results in that only the positive variations of spectral content are taken into account. We always use Equation 2.24 when refering to spectral flux later on.

Multi-Band Spectral Flux

A more advanced function that is used in Chapter 5 is a multi-dimensional one where the spectrum is divided into different frequency bands and a spectral flux is computed for each band b.

$$SF(n,b) = \sum_{k=0}^{N-1} H(|S(n,k)| - |S(n-1,k)|)\ \Omega(b,k) \tag{2.28}$$

where $\Omega(b,k)$ is the b^{th} band-pass "filter". It is a weighting function akin to the windowing functions used in the time domain. There are numerous shapes of the weighting function and spacing of the bands in the literature. In Section 5.4.3 we use a filter bank similar to the one illustrated in Figure A.1. Each $\Omega(b,k)$ has a triangular shape inside the range of frequencies of band b and is zeroed outside. The frequency bands are spaced according to the mel-frequency distribution and each band starts and ends in the middle of the previous and next bands. A more complete description of the mel-scaled filter bank and its design can be found in Appendix A.

Transients (i.e. significant peaks) are detected separately in each band and the information obtained is recombined. Usually this recombination takes into account the duplicate detection of a same transient. For instance, the same transient detected at slightly different positions in different bands is detected as such and merged back as a single event. Besides, some approaches count the number of time a transient is detected across frequency bands, and consider it to be an actual transient if it is found, for instance, in more than M distinct bands. This multi-band approach is much more robust than the two previous ones especially in noisy environments as we demonstrate in Chapter 5. Therefore, it is the method that is used for transient detection in noisy sports such as football, rugby, basketball, etc.

2.6.3 Peak Picking

Identifying what is and what is not a significant peak is a subjective topic. In general it is addressed in the literature using either a constant or an adaptive threshold [59]. A fixed threshold is usually chosen empirically and, as Figure 2.10 shows, any peak of the detection function that is above the value

of the threshold is deemed a transient. Alternatively, adaptive thresholds can be used: the value above which a peak is a transient evolves over time, depending for instance on non-linear smoothing of the detection function or on the preceding (or following) detection of another transient whose presence psycho-acoustically masks the current peak.

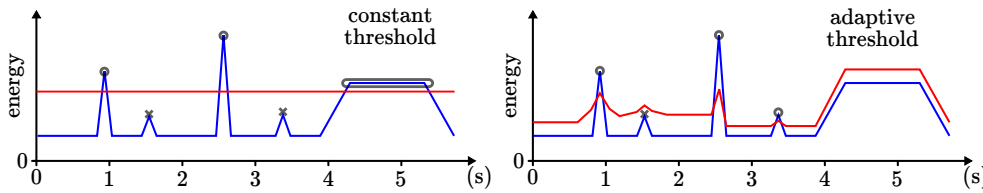

Figure 2.10. *Left: a constant threshold (red line) can miss some transient detection (gray crosses) and mistake a region with constant higher energy for a transient; right: an adaptive threshold can reduce the occurences of such mistakes, although it can still miss transients.*

Note, however, that many detection functions with so-called adaptive threshold can be rewritten as more advanced detection functions with fixed threshold. For instance an adaptive threshold proportional to the local energy of a signal and applied to the spectral flux as defined in Equation 2.24 is equivalent to a constant threshold applied to the normalized version of the spectral flux in Equation 2.25. Chapter 5 details the adaptive peak picking approaches we put in place to detect transients during sports events.

2.6.4 Processing

Once the transients are located, special care can be taken in the time-scaling process to preserve them as much as possible. Different approaches exist [58]: some split the transients and the remaining of the sound as two different signals processed separately and recombined as one output signal [54, 60, 61]; others adapt the speed factor α locally so that the transients are time-scaled with a value $\alpha = 1$ and compensate in non-transient regions so that the overall speed factor corresponds to the given constraint [62]; the same result can be obtained by adapting the frame rate locally as in [63]. In a modified phase vocoder, Röbel [64] proposes to use the original phases of the frequency bins detected as transients whereas Bonada [44] also resets the phases of all the

bins above a given frequency to the phases of the original signal. In Chapter 5 we tested two of these methods, described hereafter, to process transients in recordings of sports events.

Transient Removal

One of the methods, detailed in Section 5.4.5, is derived from [60] and is similar to a method developed in [61]. As schematized in Figure 2.11, it removes the transients from the signal by replacing them with synthetic content. The resulting transient-less signal is then time-scaled and the samples of the transient sounds are added at the appropriate position into the output using overlap-add. In our implementation, the synthetic content is filtered white noise. The spectral envelope applied by the filter to the white noise is computed as an average of the spectral envelopes of the frames surrounding the transient. Note that an SOLA-like overlap-add method can be used to insert the filtered white noise in place of the transient while reducing discontinuities.

As discussed in Section 5.4.5, this method gives the most interesting results but suffers from the drawback that the acoustic contents located before and after a transient in the input recording are mixed up in the time-scaled signal, as Figure 2.11 shows. In case the sounds located before and after a transient in the input signal are perceptually different, this causes an artifact in the output signal. Indeed, content normally located after the transient can be heard before it and content that should appear only before a transient can still be heard after it. Obviously if the contents are similar, this is imperceptible.

Local Speed Variations

The second method presented in Figure 2.12 and discussed in Section 5.4.5, is a modified implementation of the work of Masri et al. [63, 65] also described in [58]. Each time a transient is encountered, we use the last transient-less frame to time-scale the signal until the instant when the transient can be inserted at its rightful position in the output signal. Then the transient is copied from the input to the output, thus undergoing a time-scale factor $\alpha = 1$, and the first frame of sound located after the transient is used to start time-scaling again. Since each transient is inserted precisely where it ought to be in

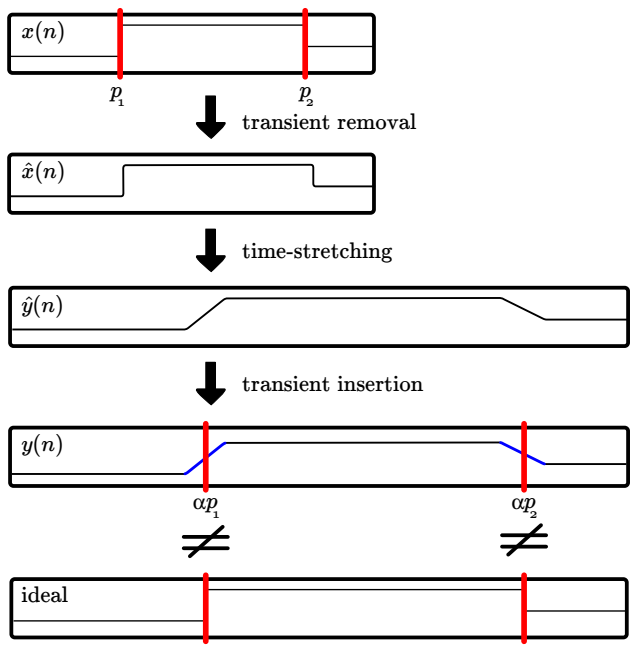

Figure 2.11. *Transients located at p_1 and p_2 are replaced with non-transient content in the original signal. The resulting signal is time-scaled by a factor α and the transients are inserted back into the output. Contrary to an ideal situation, content from after the transient influences the content before it in the output, and vice versa. Therefore the region located around the transient (blue) can sound as a disturbing artifact.*

the time-scaled signal, there is no drift of the synchronization between images and sound in sports videos, as long as every transient is properly detected.

Theoretically this method should give the best possible results. However, as shown in Section 5.4.5, it is valid only for as long as the lapse of time between two transients is large enough that the content generated has an acceptable quality. We observe that this is regularly not the case for sports recordings, especially since the frame length that we use is significantly larger than is common in the literature about time-scaling of speech and music.

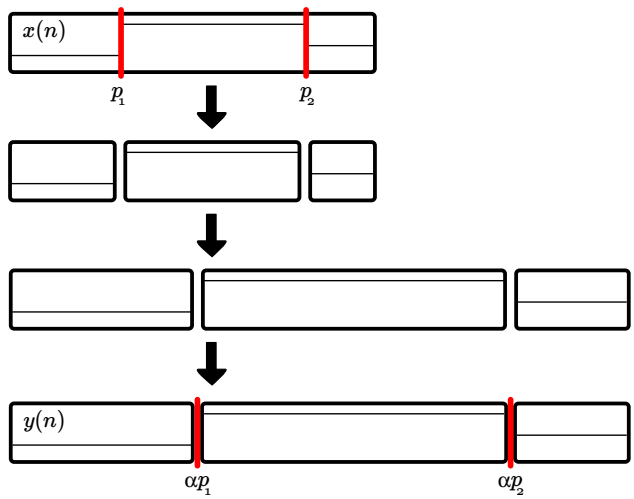

Figure 2.12. *$x(n)$ Is divided into non-transient regions which are then time-scaled separately. In the last step, transients are added back into the output signal $y(n)$. Non-transient signals are time-scaled with a speed factor slightly different from α to compensate for the fact that the transients are not time-scaled and obtain an overall speed factor equal to α.*

2.7 Sound Textures

Sound [66] or *audio* [67] *textures*[13] refer to a group of acoustical signals that present some pseudo-stationarity of their spectral and perceptual characteristics on a longer time-scale than the sinusoidal signals introduced in Section 2.1 and considered throughout Sections 2.2 to 2.6. "Longer" is usually in the order of several hundreds milliseconds or more (up to a few seconds), as opposed to the classical few dozens of milliseconds from speech and music processing.

Besides this quite comprehensive statement, there is no clear definition of what a texture is [66, 68]. The notion is typically explained through examples such as the sound of rain, fire, wind, large crowd babblings, waves, applauses, footsteps, car engines, breaking glass, etc. However, some part of the literature with stricter definition do not consider all of them as such. In the context of this thesis the broadest meaning is used as it refers to various signals, from the continuous cheers and noises of the crowd in a football stadium to the brief slide of a player's feet on the clay during a tennis tournament[14].

[13] Sometimes named *natural sounds* or *environmental sounds*.
[14] or on ice during a game of hockey, if you're that way inclined.

2.7.1 Textures Synthesis

Texture synthesis is the attempt to reproduce ad infinitum a sound texture based on a short example of it. It has applications in music composition, video games, television or movie soundtrack creation. For instance it can generate hours of applauses, possibly with variations, from a two-second example. This short example cannot be simply repeated in a loop because the human ear notices such cyclic patterns, therefore numerous methods [69] have been developed to generate natural-sounding textures based on minimal examples. They generally belong to one of two categories: granular synthesis-based or model-based.

Granular synthesis

Granular synthesis is based on Gabor's theory [70] that a sound can be decomposed into quantums of sound. These basic elements are called *atoms* or *grains* and their duration is usually between 1 and 100 milliseconds. Every sound is therefore the result of a combination, the so-called granular synthesis, of one to many grains.

Texture synthesis based on granular synthesis decomposes the example sound into its grains. The grains are then recombined either randomly [71] or according to some empirical (e.g. through statistical learning [72]) or arbitrary rules [66] or to match distance-based criterions [67, 73] or following the composer's will [74] in order to create a texture that sounds like the original but never reproduces it exactly. Note that each grain can also be processed beforehand to achieve specific audio effects.

Model-based

Model-based texture synthesis uses exactly the same principles as model-based time-scaling described in Section 2.4. The texture example is parametrized and perceptually similar sounds can be generated at will by the model. Some models, such as physical models, do not even need an example to train upon, they apply equations of sound generation and sound propagation in fluids to imitate the sound of a given physical phenomenon such as a water drop in a sink or the vibrating strings of musical instruments.

2.7.2 Time-Scaling

Some publications on texture synthesis focus on time-scaling methods. A few do so quite approximately while others, such as the work presented by Picard et al. in [73, 75], create realistic stretched versions of sound textures. We used this as one of the bases to develop a new method for time-scaling of sports recordings which is presented in Chapter 6 of this work.

The method detailed by Picard et al. decomposes a database of contact sounds recordings into non-overlapping grains and retains only those whose energy is above a given threshold to build a dictionary from these selected grains. Then for each grain of the dictionary its cross-correlation function with each recording is computed and the most significant peaks are located. As a result, for each recording, a set of peak intensities and locations associated with the different grains is created. In the final step of the algorithm, only the information for the k largest peaks is preserved for each recording, creating so-called *correlation patterns*.

This information can be used to resynthesize an approximate version of the original sound, by combining[15] the k grains, or a subset of them, at the locations t_k corresponding to the peaks. It can also be used for time-scaling, especially for time-stretching where the original signal is divided into non-overlapping grains and the grains are shifted to new positions corresponding to the stretching factor $\alpha > 1$, leaving empty spaces between each grain. These empty spaces are filled by the grains from the dictionary using the peaks from the correlation patterns that correspond to the position of each gap for that recording. Taking into account the time-scaling factor α, each time location t_k of a peak corresponds to a position αt_k in the time-scaled signal. In other words, if a gap in the extended signal covers a region that includes the sample αt_k, it means that the grain matching the peak around time t_k can be used to fill the gap, or at least part of it.

Since this method requires computation of correlation patterns over a complete database and the use of these patterns to fill the empty spaces, it is not fit for realtime live time-stretching of sounds. However, we developed a new method suitable for realtime processing of sports recordings which is also based on grain shifting. It is treated in Chapter 6 of this thesis.

[15] Generally through an overlap-add.

Chapter 3

A Phase Vocoder with Synchronized OverLap-Add

The new methods presented in this chapter are an attempt at reducing the phasiness distortion [47] already discussed in Section 2.3.2. The underlying idea comes from an experimental observation we made on the phase vocoder, using its implementation from [45], and on the phase-locked vocoder, using *Identity Phase Locking* [43] as described in Section 2.3.2 and as implemented in [23]. Our observation is that *phasiness in the vocoder does not appear (or is not perceived) immediately. It takes a few frames before becoming noticeable.*

A simple experiment to observe this phenomenon is to alter a phase-locked vocoder so that the phase-locking happens only once every C frames. The other frames are processed with a normal phase vocoder. For small values of C (typically 3 to 5 frames), the difference in phasiness with a fully locked signal is barely noticeable at all (some artifacts/ripples may appear in the spectrogram though). For larger values of C, phasiness becomes audible in the vocoder output. The loss of vertical coherence is a slow phenomenon, it is not instantaneous, and the spectral content also varies relatively slowly. Therefore, every time a peak is detected and locked its neighboring bins undergo some kind of phase reset: their final phase is only a function of the change of the peak's phase and their phase difference relatively to the peak's original phase. As for the peak, since the signal varies slowly it can be assumed that its position remains more or less coherent from one frame to another (or even across 3 to 5 frames) even if it changes of bin (the bin change is never an important jump in frequency).

Based on these observations we propose a new method that combines a time-domain and a frequency-domain approach. The method consists in a periodic reset of a phase vocoder by copying a frame directly from the input into the output and using it as a new starting point for the vocoder. The insertion point for the frame in the output is chosen by means of a cross-correlation measure, as it is done in many time-domain time-scaling algorithms.

In the following sections we present two different possible implementations for this idea. The first one, described in Section 3.1, uses an approach based on *synchronized overlap-add* (SOLA, cf. Section 2.2.3). In other words, an arbitrary frame is selected in the input and inserted at an optimal position in the output. The second implementation is discussed in Section 3.2 and is based on the same reasoning as the *waveform similarity-based synchronized overlap-add* (WSOLA, cf. Section 2.2.4). This means that an optimal frame is picked in the input and inserted at an arbitrary position in the output. The two methods are named respectively PVSOLA and PWSOLA for they combine a phase vocoder (PV) with either SOLA or WSOLA.

The present chapter, especially Section 3.1, is based on work we presented in September 2011 at the *14th Digital Audio Effects Conference (DAFx-11)* [50] and for which we obtained the *Best Student Paper Award – Bronze*. It should be noted that, in 2012, at the *15th Digital Audio Effects Conference (DAFx-12)*, Kraft et al. published a paper [51] proposing several interesting and meaningful improvements to our method, in particular for polyphonic signals. These improvements are mentioned in the informal discussions of Section 3.1.3.

3.1 PVSOLA

The *phase vocoder with synchronized overlap-add* (PVSOLA) is a time-scaling algorithm combining time-domain and frequency-domain methods. It is akin to a phase-locked vocoder whose phase locking consists only to reset, on a regular basis (every 3 or 4 frames for instance), the phase of the vocoder to that of the original signal.

3.1.1 Implementation details

We propose the following framework: first we generate C synthesis frames $(f_0, \ldots, f_{c-1})$ using one of the phase vocoder implementations described in Section 2.3.1. Each frame f_i is L-sample long and is inserted in the output signal by overlap-add at sample t_i with:

$$t_i = iR_s \tag{3.1}$$

where t_i is the position at which the first sample of the synthesis frame is inserted and R_s is the hopsize used for synthesis. We choose $R_s = L/4$, as is often done in the literature. Note that it is the largest possible value for R_s that ensures that the Hann functions, used to window each frame twice[1], overlap-add to a constant amplitude. The last frame generated (f_{c-1}) is inserted at position t_{c-1} and the next one (f_c) would be inserted at t_c. However, at t_c, instead of another vocoded frame we want to insert a frame f^* extracted directly from the input audio in order to naturally reset the phase of the vocoder but we know[2] that this would cause phase discontinuities between the partials already synthesized in the output and those of the frame f^*.

In order to minimize such discontinuities we allow the position of f^* to be shifted around t_c in the range $t_c \pm T$ (T is called the *tolerance*). The optimal shift δ is obtained by computing the cross-correlation between the samples already in the output and the samples of f^*. However, some samples of the output are "incomplete". They still need to be overlap-added with samples that would have been generated in the next steps of the phase vocoder. In other words, the samples obtained by overlap-adding the next frames $(f_c, f_{c+1}, \ldots)$ in a normally operating phase vocoder.

As a result, a frame overlap-added in another position than t_c would cause a variation in the otherwise constant[3] time-envelope of the accumulated windowing functions of the time-scaled signal. Besides, the cross-correlation would be biased toward negative shifts around t_c. To overcome these problems additional frames $(f_c, f_{c+1}, \ldots, f_{\mathrm{F}})$ are generated by the phase vocoder and temporarily inserted so that t_{F} is the smallest insertion point that respects the

[1] Once at analysis and once at synthesis, hence equivalent to a squared Hann window.
[2] See Section 2.2.3.
[3] cf. Figure 1.9.

constraint in Equation 3.2:

$$t_{\mathrm{F}} > t_c + L + T \tag{3.2}$$

which means that the first sample of the coming frame f_{F} would be inserted at least T samples after the end of f_c and that the output signal is "complete" up to sample t_{F}. In other words, no samples would be overlap-added anymore before t_{F} in a normal phase vocoder.

Positioning of the reset frame f^*

Position t_c corresponds to a position u_c in the input signal:

$$u_c = \frac{t_c}{\alpha} \tag{3.3}$$

with α the speed factor. As previously explained in Chapter 2, $\alpha > 1$ corresponds to an extension of the signal whereas $\alpha < 1$ accelerates it.

The next step consists in selecting a frame f^* of length L starting at sample u_c[4] in the input signal and adding it in the output signal at position $t_c + \delta$ with $-T \leq \delta \leq T$. We arbitrarily fix the value of the tolerance to $T = 2R_s$. Equation 3.6 defines χ, a cross-correlation measure between the frame f^* (Equation 3.4) and the output samples o (Equation 3.5) already generated:

$$f^*(n) = \{x(u_c)h^2(0), \ldots, x(u_c + L - 1)h^2(L-1)\} \tag{3.4}$$

$$o(n) = \{y(t_c - T), \ldots, y(t_c + L - 1 + T)\} \tag{3.5}$$

$$\chi = o(n) \star f^*(n) \tag{3.6}$$

where $\{\}$ denotes a frame, $h^2(n)$ is the square of a Hann window, as defined in Equation 1.4, and $\star$ is the cross-correlation operator discussed in Section 1.4. $x(n)$ and $y(n)$ are the original and time-stretched signal respectively, with $y(n)$ containing all the overlap-added synthesis frames until and including f_F. The optimal value of δ corresponds to the position of the maximum peak of $|\chi_s|$, the subset of χ, as defined in Equation 3.8, that corresponds to an insertion of f^* in the position range $t_c \pm T$. Figure 3.1 shows an example of finding the offset δ using Equations 3.7 to 3.10:

[4] rounded to the nearest integer

$$\varepsilon = L + 2T \tag{3.7}$$
$$\chi_s = \{\chi(\varepsilon), \ldots, \chi(\varepsilon + 2T)\} \tag{3.8}$$
$$p = \arg\ \text{maxpeak}(|\chi_s|) \tag{3.9}$$
$$\delta = p - T \tag{3.10}$$

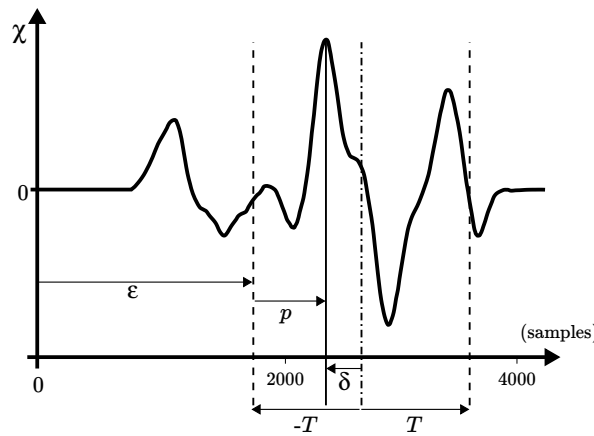

Figure 3.1. *δ is computed from the position p of the maximum peak value of a subset of $|\chi|$. The dashed lines delimit the subset χ_s and the dash-dotted line represents a positioning of f^* exactly at $t = t_c$. In this example δ is < 0 and $\chi_s(p) > 0$. The frame length L is 1024.*

Insertion of the reset frame f^*

Each frame processed through the phase vocoder undergoes two successive Hann windowings: one before the DFT and one after the IDFT before being overlap-added in the time-stretched signal. Consequently f^* also has to be windowed by the square of a Hann window, as Equation 3.4 shows, in order to overlap-add properly with the output signal and the future frames.

Then f^* is multiplied by the sign of $\chi_s(p)$. Indeed, in case the maximum peak in $|\chi_s|$ corresponds to a negative peak in χ_s, the samples of f^* are "anti-correlated" to the samples in the output signal. In other words, the samples of $-f^*$, with a time shift δ, have the best correlation with the output signal. Hence it is the frame $-f^*$ that is overlap-added to the output audio.

Before inserting f^*, the output samples between $t_c + \delta$ and $t_c + \delta + L - 1$ are windowed by a function $w(n)$ so that the overall accumulated windowing of the output remains constant, taking into account the frames yet to come. This also means that the samples of the output signal beyond $t_c + \delta + L - R_s$ that have been generated to compute the cross-correlation are set to zero. The insertion of f^* is schematized in Figure 3.2 and the computation of the envelope $w(n)$ applied to the time-stretched signal is presented in Figure 3.3 and Equation 3.11:

$$w(n) = h^2(n + 3R_s) + h^2(n + 2R_s) + h^2(n + R_s) \tag{3.11}$$

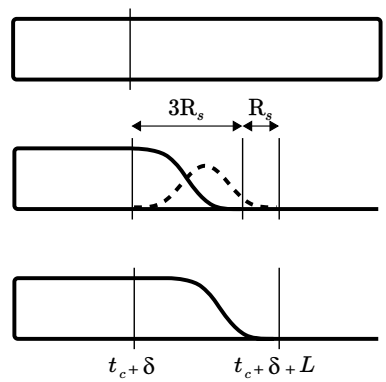

Figure 3.2. *Schematic view of the insertion of a frame f^* at position $t_c + \delta$. Top: output signal after insertion of additional frames for cross-correlation computation. Middle: windowed output signal (solid line) and frame f^* windowed by the square of a Hann window (dashed line). Bottom: resulting signal before the next iteration. The upcoming windowed frames will add to a constant time-envelope with this signal.*

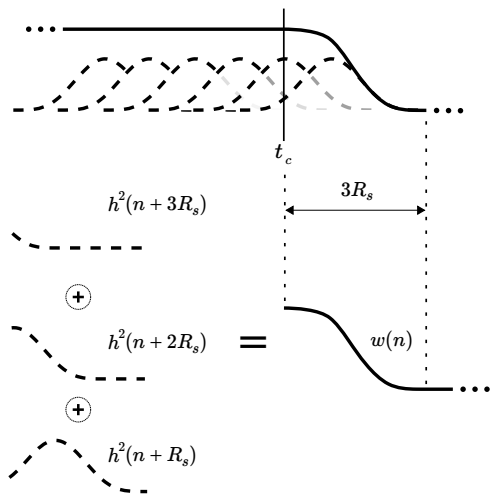

Figure 3.3. *Schematic view of the computation process for the weighting function $w(n)$ that will be applied to the output signal after $t_c + \delta$. Top: in a standard phase vocoder, the squared Hann windows would sum to a constant value except for the last samples because there are frames not yet overlap-added after t_c. We want to reproduce that behavior at $t_c + \delta$ so that f^* overlap-adds seamlessly. Bottom: The time envelope is the sum of three squared Hann windows with a shift R_s between each one.*

Finally, since frame f^* has been inserted "as is", the phase vocoder can be reinitialized to start a new step of the time-scaling process as if f^* were its initial frame f_0 and $t_c + \delta$ were its initial time position t_0. Note that each analysis frame used during this new step must be inverted if $\chi_s(p) < 0$ since the frame overlap-added to the output is $-f^*$ instead of f^*.

3.1.2 Discussion

It is important to notice that due to the accumulation of shifts δ (one for each iteration) a drift from the original speed factor α could occur if no measure is taken to correct it. In our implementation we sum the values of δ for each phase reset and obtain a drift Δ. When Δ exceeds $\pm R_s$ the number of frames

synthesized in the next iteration will be $C \mp 1$ and the value of Δ will change to $\Delta \mp R_s$. Theoretically Δ could even exceed $\pm 2R_s$, in which case the number of frames synthesized will be $C \mp 2$ and Δ is changed by $\mp 2R_s$.

Another interesting fact is that if we set $C = 0$, the resulting algorithm is close to a SOLA-like method except that the additional frames used for the cross-correlation are still generated by a phase vocoder. On the contrary, $C = \infty$ changes the method back into a non-locked standard phase vocoder.

In our publication [50], we used the maximum of the cross-correlation subset χ_s, as opposed to its maximum *peak*, to determine the optimal shift δ. However, we noticed later that regularly[5] the maximum corresponds to a value of $\delta = -2R_s$. In other words, the first sample of χ_s is the maximum value and it is a sample from a slope of the correlation coming down from a peak in the part of χ that we do not use (i.e. a peak in $\{\chi(0), \ldots, \chi(\varepsilon - 1)\}$). Logically, in voiced part of the signal, that peak is more optimal to use for the value of δ as it corresponds to a synchronization point of the phases of the signal. Two approaches are possible to fix this: either allow the algorithm to pick δ outside of the range $\pm 2R_s$ when this particular issue occurs, or force the position selected by the algorithm to be a *peak* inside χ_s. In any case, this is a rare event as most of the maximums selected were already not at $\delta = -2R_s$, and thus are actually peaks. Always using a peak to deduce δ instead of an overall maximum simply removed some of the phase discontinuities that could be heard in our original implementation hence improving the acoustic quality.

Finally, in Section 3.1.1 we consider the first sample of a frame as the reference for positioning. One might use the middle sample of each frame instead. This will not create any significant difference with the method proposed above.

3.1.3 Results

For the following tests we implemented a modified version of the algorithm [45] based on frame generation and described in Section 2.3.1. We performed both formal and informal assessments as presented hereafter.

[5] Often enough that we noticed the problem.

Formal listening tests

We use sentences selected from the CMU ARCTIC databases [76] among the four US speakers, namely *clb*, *slt*, *bdl* and *rms* (two female and two male speakers). Fifty sentences are randomly picked for each speaker and each sentence is processed by four different algorithms: a phase-vocoder, a phase-locked vocoder, a time-domain method (SOLAFS) and our method PVSOLA. Each process is applied with two speed factors: $\alpha = 1.5$ and $\alpha = 3$ (i.e. 1.5 and 3 times slower).

For the two phase vocoders we use the implementations available in [23] and for SOLAFS we use the implementation from [77]. We empirically set $L = 512$ samples and $R_s = L/4$ for the vocoders and PVSOLA. In our informal tests SOLAFS generally provided better quality with $L = 256$ so we kept that value. The parameters specific to PVSOLA are $C = 3$ and $T = 2R_s$.

PVSOLA is compared to the other three methods via a *Comparative Mean Opinion Score* (CMOS) test [78]. Participants are given the unprocessed audio signal as a reference (R) and they are asked to score the comparative quality of two time-stretched versions of the signal (both of them with the same speed modification). One is PVSOLA, the other is randomly chosen among the three state-of-the-art algorithms. The two signals are randomly presented as A and B. Each listener takes 30 tests, 10 for each concurrent method. The question asked is: "*When compared to reference R, A is:* much better, better, slightly better, about the same, slightly worse, worse, much worse *than B ?*"

Each choice made by a listener corresponds to a score between ± 3. In case A is PVSOLA, "much better" is worth 3 points, "better" 2 points and so on until "much worse" which means -3 points. On the contrary when B is PVSOLA, the scale is reversed with "much worse" worth 3 points and "much better" -3 points. In short when PVSOLA is preferred it gets a positive grade and when it is not it gets a negative one. Sixteen people took the test (among which 9 are working in speech processing) and the results are shown in Table 3.1 and Figure 3.4.

From these results one can see that for a speed slowdown factor of 1.5 our method is globally preferred except for SOLAFS with female voices where both methods are deemed equivalent. Besides, SOLAFS performs relatively

Table 3.1. *CMOS test results with 0.95 confidence intervals for female (*clb *and* slt*) and male (*bdl *and* rms*) speakers. PVSOLA is compared to the phase vocoder (*pvoc*), the phase-locked vocoder (*plock*) and* SOLAFS.

	female	
α	1.5	3
pvoc	2.03 ± 0.3	0.66 ± 0.43
plock	0.97 ± 0.41	1.86 ± 0.3
solafs	0.14 ± 0.32	1.21 ± 0.27

	male	
α	1.5	3
pvoc	2.49 ± 0.32	1.05 ± 0.47
plock	1.78 ± 0.29	1.71 ± 0.3
solafs	1.13 ± 0.36	1.77 ± 0.27

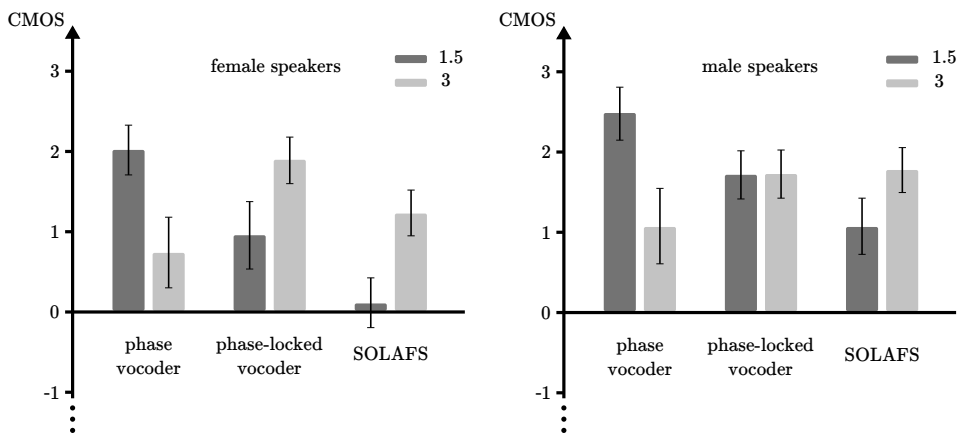

Figure 3.4. *Results for the CMOS test for female speakers* clb *and* slt *(left) and male* bdl *and* rms *speakers (right). The dark and light gray bars represent the mean CMOS score for a speed ratio of respectively 1.5 and 3. 0.95 confidence intervals are indicated for information.*

better than the phase-locked vocoder which in turn performs better than the phase vocoder. This is an expected result as time-domain methods usually give better results when applied to speech and the phase-locked vocoder is supposed to be better than the phase vocoder.

For the higher slowdown factor 3, our method is again observed to outperform other approaches, notably better than SOLAFS and the phase-locked vocoder in both tables, but it has lost ground to the normal phase vocoder which has a better score than the two other approaches. After the test we discussed this with the listeners and we could establish that it was not a mistake. Indeed, with this time-stretching ratio every method produces more artifacts (frame repetition for SOLAFS, metallic sound for the phase-locked vocoder, phasiness for the phase vocoder and some sort of amplitude modulations for PVSOLA). The listeners said that in some cases they "preferred" the defect of the phase vocoder to that of PVSOLA for a certain number of sentences of the dataset. It is still a minority of files for which this happens and the overall result remains in favor of PVSOLA but this is too significant to be simply shrugged off.

At the time, these tests were run with the original implementation using absolute maximums of the cross-correlation. Later on, we ran the same listening tests again, but using absolute maximum peaks instead, and obtained very similar results presented in Table 3.2. Therefore, these new results do not point to a significant improvement of our method when compared to state of the art. However, informal comparisons of the two PVSOLA implementations by experts in audio processing logically confirmed that using correlation peaks reduced the amount of phase discontinuities without introducing new artifacts.

Table 3.2. *CMOS test results with 0.95 confidence intervals for female and male speakers. This time, contrary to the comparison in Table 3.1, PVSOLA uses the absolute maximum peak of correlation.*

	female	
α	1.5	3
pvoc	2.09 ± 0.3	1.29 ± 0.34
plock	0.87 ± 0.3	1.29 ± 0.34
solafs	0.92 ± 0.39	1.35 ± 0.3

	male	
α	1.5	3
pvoc	1.95 ± 0.33	1.57 ± 0.3
plock	1.77 ± 0.3	1.73 ± 0.3
solafs	1.08 ± 0.28	1.41 ± 0.32

Informal tests and discussions

Several values for C and L have been tried and the best trade-off for harmonic signals seems to be $C = 3$ and $L = 512$ samples for a sampling frequency $F_s = 16$ kHz (i.e. $L = 32$ ms). As for other sampling frequencies (in singing and music data) we set L so that it also corresponds to about 30 ms. Nevertheless we noticed that in general the algorithm is not overly sensitive to the value of L (between 20 and 40 ms). For $C = 3$ and a reasonable speed factor (between 1 and 3 times slower, $1 \leq \alpha \leq 3$) we generally notice an important reduction of the phasiness compared to the phase vocoder. We generated some test samples for even slower speed factors ($\alpha = 5$) with mixed results (some good, others presenting many artifacts).

For values of $C \geq 5$, perceptible phase incoherencies appear in the time-stretched signals probably because the phases of the different partials are out-of-phase with each other. It seems that the cross-correlation measure can help to match some of these partials with the ones from the input frame f^* but not all of them, thereby creating artifacts that resemble an amplitude modulation (the audio sounds "hashed", sometimes a beat appears at a frequency corresponding to CR_s). Note that even for values of $C \leq 3$ these mismatches may still appear but to a lesser extent, they are often almost inaudible. However, discussions with listeners have shown that in some worst-case scenarios they can become a real inconvenience as explained in the previous section.

We applied the algorithm to various signals: speech, singing voice, mono and polyphonic music and obtained improved results over all other methods for harmonic signals (speech, singing and some music). One of the notable advantage of the process when applied on speech signals is that it preserves their shape-invariance property [79]. As for polyphonic signals, the algorithm suffers from audible phase mismatches that even small values of C cannot fix. This issue is tackled by Kraft et al. in [51] by tracking sinusoidal peaks in F^*, the DFT of f^*, in the same way identity or scaled phase locking do in Section 2.3.2. Then the shift δ is computed and the phases of all the non-peak components of F^* are modified through phase vocoding, according to that shift, while the phases of the detected peaks (and surrounding bins) are left untouched. Eventually, a modified frame is obtained through inverse DFT of the modified F^* and is inserted in the output signal. This ensures that the phases of the detected dominant partials, which are more likely to be correctly synchronized by the cross-correlation and thus by the shift δ, are reset properly

while the other components remain in phase with the output signal thanks to the phase vocoder, hence reducing the risk of amplitude modulations due to phase discontinuities. This may also improve the precision of the shift δ if the correlation is computed only for the detected partials.

As a side-effect of the algorithm, transients tend to be well-preserved contrary to what happens with time-domain (transient duplication) or phase vocoder-based algorithms (transient smearing). We argue that f^* can be advantageously positioned so that the transient is preserved due to the relatively large value of T. Also Kraft et al. [51] suggest that the regular and frequent copy of frames directly from the input signal makes it likely that a transient is included in one of these frames and thus reproduced exactly in the output signal, but that it also makes transient duplicate as probable.

The main drawback of our method lies in its computational complexity when compared with time-domain or phase vocoder approaches. Indeed, not only do we compute a cross-correlation every C frame but we also generate extra frames for its computation that will be dropped eventually and replaced by new ones. Roughly speaking we measured that our MATLAB implementation was three to four times slower than a phase vocoder. A profiling of the process shows that the most time-consuming task is by far the cross-correlation computation (about 40%). However, results of benchmarking within MATLAB must be considered with care since some operations (such as selecting a frame in a signal) are not well-optimized. We estimate that a C implementation of PVSOLA could be less than two times slower than that of a phase vocoder.

In [51], Kraft et al. address this issue by computing the cross-correlation only between the last output frame f_c of the vocoder and the input frame f^* to be inserted. This cross-correlation presents a bias, because f_c is windowed, but it is compensated by weighting the cross-correlation with the inverse of the auto-correlation of the windowing function[6]. Besides, in order to compensate for the accumulated drift Δ, the cross-correlation is further weighted to introduce a bias in the shift δ so that it tends to bring Δ back to 0. Finally the output signal is divided[7] by the global envelope $w(n)$ in the region $[t_c, \ldots, t_c+3R_s]$ to cancel out its influence and it is multiplied by the same envelope $w(n)$ in the region $[t_c+\delta, \ldots, t_c+\delta+3R_s]$ so that f^* can be overlap-added while preserving the signal envelope.

[6] limited to the part where $w(n) > 10^{-3}$ to avoid rounding errors and divisions by zero.
[7] Once again this is limited to its central part to reduce rounding errors

3.2 PWSOLA

The *phase vocoder with waveform similarity-based synchronized overlap-add* (PWSOLA) is a time-scaling algorithm combining time-domain and frequency-domain methods. It uses the same underlying principle as PVSOLA and periodically resets the phases of the vocoder to those of the original signal by inserting frames from said signal. Contrary to the previous method though, the position at which a frame is inserted is fixed. It is not adapted according to some correlation measurement. Instead, the input frame is selected so as to maximize that correlation, the same way it is achieved in methods such as WSOLA [33] or SOLAFS [26].

3.2.1 Implementation details

We propose a framework similar to the one described in Section 3.1: first we generate C synthesis frames $(f_0, \ldots, f_{c-1})$ using a phase vocoder. Each frame f_i is L-sample long and is inserted in the output signal by overlap-add at sample t_i with:

$$t_i = iR_s \tag{3.12}$$

where t_i is the position at which the first sample of the synthesis frame is inserted and R_s is the hopsize used for synthesis. We choose $R_s = L/4$ as already mentioned in PVSOLA. The last frame generated (f_{c-1}) is inserted at position t_{c-1}, the next one (f_c) should be inserted at t_c. Now instead of another vocoded frame we want to insert a frame f^* extracted directly from the input audio in order to naturally reset the phase of the vocoder.

However, if f^* is chosen arbitrarily it is likely that it will cause phase discontinuities between the partials already synthesized in the output and those of the frame f^*. Instead, we propose to select a frame from the input that maximizes a correlation measure with the samples of the output signal located after t_c.

Selection of the reset frame f^*

As in PVSOLA, t_c corresponds to a position $u_c = {}^{t_c}/_{\alpha}$ in the input signal, with α the speed factor. Therefore, f^* corresponds to samples $\{x(u_c+\delta), \ldots, x(u_c+$

$L-1+\delta)\}$[8] with $-T \leq \delta \leq T$ and T the *tolerance*. If we used PVSOLA boundaries backwards, the tolerance would be $T = \pm 2R_s/\alpha$. However, for large values of α (e.g. $\alpha > 3$), this would significantly reduce the number of possible frames. Likewise, small values of α would increase unreasonably the range of samples used for the search, although this is less important to us, since we are focusing mainly on slow motion. We arbitrarily fixed $T = R_s$.

Equation 3.15 defines χ, a cross-correlation measure between the input samples in which the frame lookup happens (Equation 3.13) and the output samples o (Equation 3.14) already generated:

$$\iota(n) = \{x(u_c - T), \ldots, x(u_c + L - 1 + T)\} \tag{3.13}$$
$$o(n) = \{y(t_c), \ldots, y(t_c + 3R_s - 1)\} \tag{3.14}$$
$$\chi = o(n) \star \iota(n) \tag{3.15}$$

The optimal value of δ corresponds to the position of the maximum peak of $|\chi_s|$, the subset of χ, as defined in Equation 3.17, that corresponds to a selection of the first sample of f^* in the position range $u_c \pm T$.

$$\varepsilon = L \tag{3.16}$$
$$\chi_s = \{\chi(\varepsilon), \ldots, \chi(\varepsilon + 2T)\} \tag{3.17}$$
$$p = \arg\ \text{maxpeak}(|\chi_s|) \tag{3.18}$$
$$\delta = T - p \tag{3.19}$$
$$f^* = \{x(u_c + \delta), \ldots, x(u_c + L - 1 + \delta)\} \tag{3.20}$$

Exactly like in PVSOLA, f^* has to be windowed by the square of a Hann window, defined in Equation 1.4, before being overlap-added so that the overall signal amplitude is preserved. Also if $\chi_s(p)$ is negative, f^* is changed to $-f^*$ before insertion and every frame generated by the vocoder during the next iteration has to be inverted as well.

[8] with u_c rounded to the nearest integer

3.2.2 Discussion

Contrary to PVSOLA, no further step is required before inserting f^*. The output is already windowed "naturally" so that the frame overlaps seamlessly, and the phase vocoder can be immediately reset to start a new cycle of frame generation. This makes the algorithm much faster than PVSOLA since no extra frames have been computed for the cross-correlation computation, which was one of the main drawbacks of PVSOLA.

Besides, there is no drift Δ since each frame f^* is overlap-added exactly at sample t_c. However, we think that this is also the cause of the major defect of this method. Indeed, the output signal generally presents an audible periodic amplitude modulation whose period is equal to CR_s samples. We attribute this pulse to the small phase discontinuities that happen[9] each time a reset frame f^* is inserted and to the fact that the auditory system is sensitive to such a regular pattern, absent from the PVSOLA algorithm.

For this reason we did not consider this algorithm for formal testings since its quality is so obviously and objectively flawed.

3.3 Conclusions

PVSOLA is a new method for time-scaling that combines time-domain and frequency domain approaches. It consists in a periodic reset of a phase vocoder by copying a frame directly from the input into the output and using it as a new starting point for the phase vocoder. The insertion point for the frame in the output is chosen by means of a cross-correlation measure. Informal listening tests have highlighted a reduction of the phase vocoder's phasiness and formal listening tests have shown that our method was generally preferred to existing related state-of-the-art algorithms. Both formal and informal tests have pointed out that under certain circumstances the quality of the time-stretched audio could be perceived poorly because of phase discontinuities in the signal. However, using maximum peaks of the correlation instead of absolute maximums to choose the insertion position of each reset frame effectively reduces the amount of discontinuities.

[9] Remember that the use of a cross-correlation measure is an attempt at minimizing the discontinuities when inserting f^*, but it does not suppress them completely.

Part II

Audio Time-Scaling for Slow Motion Sports Videos

Chapter 4

Database and Tools

EVS Broadcast Equipment provides its clients with hardware and sofware tools to store, manage, process and broadcast thousands of hours of video streams but, of course, EVS does not own any of the content. However, with permission of the copyright owners, they made available for us dozens of short excerpts of several sports of interest. The audio content of these videos is used in Chapters 5 and 6 to assess the results of different time-scaling methods.

The current chapter describes this dataset in Section 4.1 in terms of content, quality, duration and so on. EVS also provided us with video players, able to play the proprietary MXF format of the sports videos, and a software library with which we developed small utilities in order to extract the audio streams from the original videos and, once processed, encapsulate them in slow motion videos. All these tools are briefly presented in Section 4.2. Finally we describe the manual annotations we made of some of the recordings in Section 4.3.

4.1 Recordings

Most of the original recordings come as *Material eXchange Format* (MXF) videos. However, some are simple audio files. All the recordings are sampled at 48 kHz for the sound and either 25 or 30 fps for the video. Their content is unevenly distributed among nine different sports as follows:

- **football**: 103 excerpts coming from 5 different games,
- **rugby**: 30 excerpts of one game,

- **cricket**: 25 excerpts of one game,
- **ice hockey**: 18 excerpts of one game,
- **tennis**: 17 excerpts of one game on clay,
- **basketball**: 16 excerpts of one game,
- **baseball**: 12 excerpts of one game,
- **car race**: 7 excerpts,
- **athletics**: 1 complete recording of a 110 metres hurdles race.

Figure 4.1 contains spectrograms representative of some of the sports contained in the database, to illustrate the variety of content at our disposal.

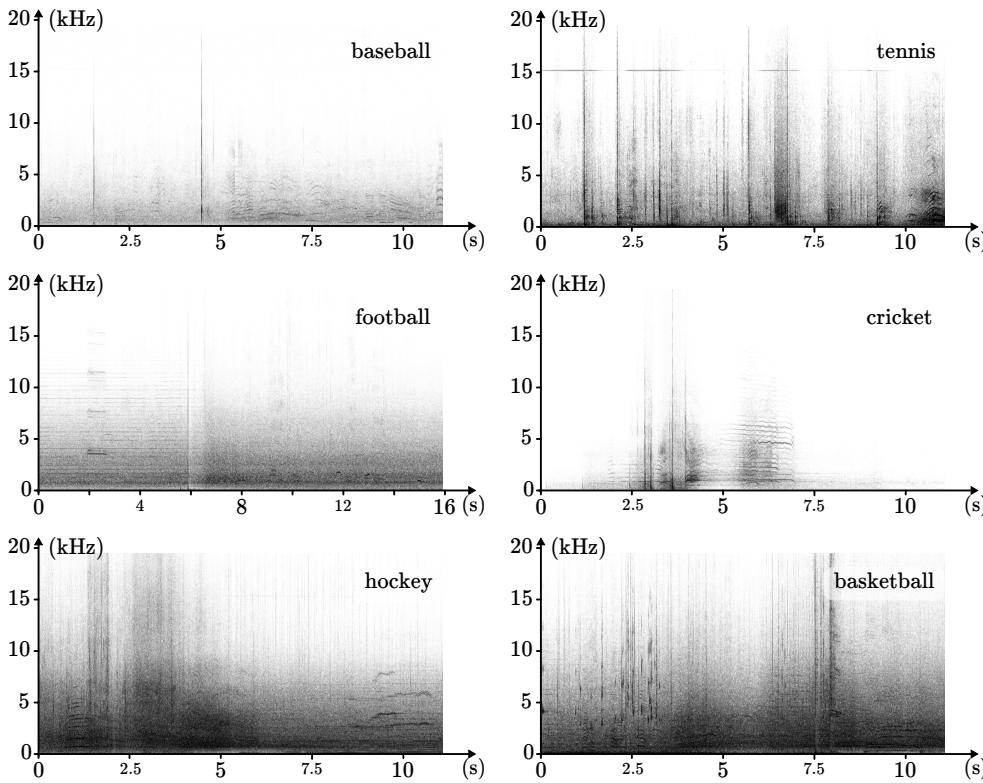

Figure 4.1. *Examples of spectrograms extracted from the database.*

Note that some files are duplicates of each other with only the viewpoint changing but not the audio, especially in football, as illustrated in Figure 4.2. It is also important to know that the recordings have generally been created by sound directors during the events, through professional audio mixing of one or several sources. These are the sounds intended for live broadcast to viewers and, as such, they are the audio signals that we want to time-scale. The raw sounds coming directly from the on-field microphones are not available.

4.1.1 Football

Football is the first and main sport on which the different methods of time-scaling have been tested and demonstrated during the research. It is also the sport for which we have the most files at our disposal. This is because football is the most popular sport with the largest potential audience worldwide. It is also the first market that has shown concrete interest for adding audio to slow motion videos.

The 103 files of the database belong to five different events, namely:

- 48 files from a match between Barcelona and Real Madrid on December 13$^{\text{th}}$, 2008, 2008 – 2009 La Liga season, (*BRM*),
- 11 files from the Brazil – Argentina final of the 2005 FIFA Confederations Cup (*BA*),
- 6 files from the 2007 Champions League Final between A.C. Milan and Liverpool (*CL*),
- 32 files from a match between Saint-Étienne and Olympique Lyonnais, February 12$^{\text{th}}$, 2011, 2010 – 2011 Ligue 1 season (*SEOL*),
- 6 files from an unindentified match during the 2010 World Cup in South Africa, as test cases for the now infamous vuvuzelas (*VV*).

However, for some of these events, only a part of the files are actual unique recordings. The others are duplicates recorded with another camera but a common audio content as illustrated with the penalty of Figure 4.2. These duplicates can be useful when working on image/sound synchronization since some events can be heard while not being seen from a given angle but are visible from a different one. In the case of BRM's 48 there are actually only

17 different audio recordings, and 8 among SEOL's 32, for a grand total of 48 different audio files among the five matches.

Figure 4.2. *The same football action, a penalty, as seen in two different MXF files corresponding to the same identification code, "548", from the BRM subset.*

Most of the files are encapsulated in an MXF video container with audio as 4-channel 48kHz 24-bit/PCM waveforms without compression. Depending on the event, the first two channels correspond either to the live sound of the players, the stadium and the referees (ambient sounds) or to the voices of the television commentators, which we never used, while the next two channels always contain the ambient sounds. The six VV files are mono WAVE files with-

out images. They are recorded in 16-bit/PCM and sampled at 48kHz. Note that all the 24-bit/PCM recordings were requantized down to 16-bit/PCM by simple truncation of the 8 less significant bits, without dithering. The level of noise of the recordings makes this truncation imperceptible. Besides, the methods described in the following chapters can be applied to any type of audio encoding regardless.

The processing of stereophonic or polyphonic sounds with time-scaling algorithms is a whole problem in itself because any loss of synchronization between the channels not only destroys the spatialization effect but also creates undesirable echoes. Besides, in every stereo recording at our disposal the two channels are either identical ("fake" stereo) or all but the same. Therefore, we always worked on the first channel of the file that corresponds to ambient sounds (i.e. first or third channel depending on the recording).

Finally, in order to perform the tests detailed in Chapters 5 and 6, we selected some relevant examples among the database. Mostly four files are used, one from BA and three from BRM. The file from BA, identified by the code 625F, has been selected because it features a lot of drums and applauses as well as several whistling sounds from the audience and the referee. The three files from BRM, identified by the codes 548, 549 and 554, have been chosen because they contain significant acoustic events happening during interesting passages of play which are visually meaningful for public demonstrations: a penalty (shown in Figure 4.2), a dribble/shoot/save action and a goal.

Considering this, the 48 files from the complete test set account for a total of 67,213,488 samples, or 1,400.281 seconds of recordings, whereas the four files (625F, 548, 549 and 554) used during the development stage contain respectively 2,944,320, 1,532,160, 1,269,120 and 2,382,720 samples or 61.34, 31.92, 26.44 and 49.64 seconds.

4.1.2 Rugby

The 30 rugby files come from a single event, a match between Wales and Italy during the 2008 Six Nations championship, February 23rd. Like the football recordings, they are encapsulated as MXF video containers with audio as 4-channel 48kHz 24-bit/PCM waveforms without compression, the first two channels correspond to the voices of the television commentators which we

never used while the next two channels are the live sound of the players, the stadium and the referees. All the recordings are truncated to 16-bit/PCM and only the third channel is used during the tests.

Among the 30 files, only 10 are unique, the others are duplicates from different camera angles. Besides, five of these 10 recording are subsets of the other five which brings the amount of original content down to five files. There is a total of 6,664,320 samples in these five files which corresponds to 138.84 seconds of recording.

The acoustical content of these files is similar to the content of the football recordings (big stadium, thousands of people in the crowd, whistles, impact sounds with the ball, etc.) and it became obvious from some informal experiments that any observation made for the football recordings applied directly to these rugby recordings as well. Therefore, these files have not been used intensively during the research, only occasionally as a validation set.

4.1.3 Cricket

The 25 cricket files come from a single event whose competitors are unkown. They are encapsulated as MXF video containers with audio as 4-channel 48kHz 24-bit/PCM waveforms without compression. As explained in the previous sections, the audio samples are truncated to 16-bit/PCM. As for the channels, the first one contains the voices of the commentators and is not used, the second channel correspond to a microphone positionned behind the batsman and the third and fourth channels are the recordings of the stadium ambience. Note that, contrary to previous cases, channel 3 and 4 can present significant differences.

The second channel is the most important for cricket specialists as it provides the sound of the impacts of the ball, whether it is on the cricket bat or the wickets behind the batsman. It allows them to deduce instantly several information such as whether the ball touched the bat or the wickets, the position of the impact on the bat and the direction the ball takes afterwards.

Therefore, it is important for that impact sound to be properly reproduced during a slow motion video. Besides, for slightly different reasons, it is also the most interesting channel for our research, since it contains not only the impact sounds but also the footsteps of the players as well as their talks and

cheers as opposed to the stadium recordings of the third and fourth channels which present much less diversity in their sounds.

There is a total of 29,233,920 samples or 609.04 seconds of recording in each channel of these 25 files. One file, identified by the code 130A, has been used for most of the tests throughout the research, it contains 987,840 samples or 20.58 seconds of recording in each channel. It features several footsteps, a throw and a subsequent impact, followed by cheers from the players.

4.1.4 Ice Hockey

The 18 files come from a single event, a ice hockey match between the Washington Capitals and New Jersey Devils during the 2003-2004 NHL championship. They are encapsulated as MXF video containers with audio as 4-channel 48kHz 24-bit/PCM waveforms without compression, the first two channels correspond to the voices of the television commentators which we never used while the next two channels are the live sound of the players, the stadium and the referees. All the recordings are truncated to 16-bit/PCM and only the third channel is used for the tests.

There is a total of 18,018,001 samples or 375.375 seconds of recording, but note that some of the 18 recordings overlap for a few seconds with their direct neighbors in the database, so the global figures are approximative. During the research we focused mostly on three non-overlapping files, with codes 615E, 617A and 617F. [1] The total duration of these three files is 2,596,995 samples or 54.104 seconds. The three selected files contain events such as shoot, goal, impact of the puck with a wall, impact of players with the wall, etc.

4.1.5 Tennis

The 17 files feature parts of the final match of the 2007 French Open, on clay, opposing Justine Henin to Ana Ivanovic on June 9th, 2007. They are encapsulated as MXF video containers with audio as 4-channel 48kHz 24-bit/PCM waveforms without compression. Every channel contains the live sound of the players, the stadium and the referees. All the recordings are truncated to 16-bit/PCM and only the first channel is used for the tests.

[1] Tests were also conducted with files 615B and 615C, but to a much lesser extent.

There is a total of 20,843,520 samples or 434.24 seconds of recording, but during the research we focused mostly on two files, with codes 610C and 610D. The total duration of these files is 2,208,000 samples or 46 seconds. Each recording features a complete rally with service, ball hits and bounces, footsteps, slides and screams as well as referee shouts and scores and applauses from the audience.

4.1.6 Basketball

The 16 basketball files come from a single event, a match between the Washington Wizards and Indiana Pacers during the 2004-2005 NBA championship. They are encapsulated as MXF video containers with audio as 4-channel 48kHz 24-bit/PCM waveforms without compression. The first two channels correspond to the live sound of the players, the stadium and the referees while the next two channels are the voices of the television commentators which we never used. All the recordings are truncated to 16-bit/PCM and only the first channel is used for the tests.

There is a total of 16,273,856 samples or 339.039 seconds of recording, but during the research we focused mostly on two files, with codes 618F and 619B. The total duration of these files is 1,704,103 samples or 35.502 seconds. The two selected recordings contain events such as dribbles, a slam dunk, a swish field goal, footsteps and slides, whistles, etc.

4.1.7 Baseball

The 12 baseball files come from a single unknown event. They are encapsulated as MXF video containers with audio as 4-channel 48kHz 24-bit/PCM waveforms without compression. The first two channels correspond to the live sound of the players, the stadium and the referees while the next two channels are the voices of the television commentators which we never used. All the recordings are truncated to 16-bit/PCM and only the first channel is used for the tests.

There is a total of 10,780,372 samples or 224.591 seconds of recording, but during the research we focused mostly on one file, with code 613B. The total duration of this recording is 892,892 samples or 18.602 seconds and it contains

a throw and a ball hit followed by the sound of crowd cheers and the stadium commentator who can be heard in the background. Note that the baseball recordings are generally quiet compared to all the other sports and, for instance, makes the detection of the impact sound between the ball and the bat much more reliable.

4.1.8 Car Race

The 7 files contains recordings of GT1 cars and are encapsulated within MXF video containers with four audio channels encoded as 48kHz 24-bit/PCM waveforms without compression. All four channels have the same content and correspond to the live sound either from outside the car (in the pit lane) or from inside the car. All the recordings are truncated to 16-bit/PCM and only the first channel is used for the tests.

These files have been obtained relatively late in the research. They have been used to test algorithms developed beforehand for the other sports, and mainly for demonstration purpose during the 2011 IBC Conference. However, the overall quality is not as good as with the other sports and not much time has been spent to specifically improve the results for this sport.

4.1.9 Hurdles

The file is a record of an almost complete race of 110 metres hurdles. It is encapsulated in a MXF video container with four audio channels encoded as 48kHz 24-bit/PCM waveforms without compression. The first two channels correspond to the live sound of the runners, the stadium and the stadium speaker while the next two channels are the voices of the television commentators which we never used. All the recordings are truncated to 16-bit/PCM and only the first channel is used for the tests. The file contains 622,080 samples or 12.96 seconds of recording. The first second of the race is missing and thus the starting “gunshot”.

4.2 Tools

4.2.1 MXF Library

EVS provided us with a C++ software library for Windows with which we can extract all the metadata and the audio and video streams from the MXF files. Thanks to this library, we wrote several applications:

- `mxf2audio.exe` extracts the audio content of a given channel of an MXF file into a binary file where each pair of bytes represents a sample (i.e. a file containing only the audio samples in 16-bit/PCM, without any header),
- `mxfx3.exe` creates a time-scaled MXF file from an input MXF file (for the video stream) and an input binary audio file which is the time-scaled version of the file obtained through `mxf2audio.exe`. Constant and variable speed factors are supported by the application, but, obviously, it has to match the time-scaling of the input audio otherwise image and sound will not be synchronized.
- `mxf2stdout.exe` extracts the video frames and write the 8-bit RGB values of each pixel to `stdout`. Some videos have frames with only half the number of lines of a HD video (540 instead of 1080), an option of `mxf2stdout.exe` compensates this by duplicating every line sent to `stdout`.
- `mxf2avi.exe` reads values from `stdin` as video frames whose dimensions are given as parameters and encodes them into an MPEG file. This creates video files that can be played with most video players. The speed factor of the file generated can be set as with `mxfx3.exe` to create silent slow motion videos that can then be multiplexed with time-scaled audio. This is a more practical substitute to `mxfx3.exe` to perform quick tests since generating an MXF file with `mxfx3.exe` takes up to several minutes whereas generating the equivalent MPEG file lasts only a few seconds. Besides, the MPEG files are heavily compressed which makes it easier to transfer and manipulate them.[2]

[2] A time-scaled MXF file is often larger than 1GB whereas the compressed files only takes a few MBs.

`mxf2avi.exe` does not actually depend on the MXF manipulation library and `mxf2stdout.exe` and `mxf2avi.exe` were initially meant to be a single executable. Nevertheless the extraction and the MPEG compression steps have been split because of incompatibilities between the MXF library and `FFmpeg` for Windows which is used for the MPEG encoding in `mxf2avi.exe`. The two utilities communicate through the pipe operator "`|`" as the following example shows

```
C:\> mxf2stdout.exe -i "foot-548.mxf" | mxf2avi.exe \
     -w 1920 -h 1088 -r 3 -o "foot-548-slow-motion.mpeg"
```

In this case the resulting HD (1920x1088) MPEG file is three times slower (option `-r 3`) than the original video `foot-548.mxf`. It does not have any sound yet, the time-scaled audio channel has to be added to the video in a further step that is achieved using the `ffmpeg` command-line application.

4.2.2 MXF Video Players

Besides the MXF library, EVS also provided us with three different video players, illustrated in Figure 4.3, to visualize and navigate into the original and processed MXF files.

4.3 Annotation

An effort to annotate the database was started and full annotations exist for the baseball, the rugby and the BRM subset of the football database. The tennis file 610C, which is the most used for the tests on tennis, is annotated as well.

Each annotation of an audio file consists of two text files. The first file describes the ambient sounds (applause, speech, cheers, whistles, etc.) over time and the second one contains the positions of beginning and end of each transient, with each transient that could be identified (shoot in a ball, hand claps, drum, etc.) labelled as such.

Figure 4.3. *Screen captures of the three players at our disposal, with different sports: tennis, baseball and ice hockey.*

4.3.1 Statistics

Baseball

224.591 seconds (10,780,372 samples), containing 115 transients of which 83 are identified: 10 drums, 50 isolated applauses, 18 ball hits, 4 echoes of the bat/ball impacts and one attack of speech.

Football

444.8 seconds (21,350,400 samples), containing 297 transients of which 275 are identified: 57 ball impacts, 184 drums, 33 onsets/releases of whistles or horns.

Rugby

138.84 seconds (6,664,320 samples), containing 130 transients of which 99 are identified: 78 isolated applauses, 3 ball impacts and 18 onset/releases of whistles.

Tennis

25.6 seconds (1,228,800 samples), containing 55 transients of which 43 are identified: 11 footsteps, 11 ball rebounds, 8 ball/racket impacts, 5 isolated applauses, 5 echoes of impacts, 1 slide on the clay, 1 speech onset and 1 racket hitting the ground.

Chapter 5

On the Use of Old Recipes for New Material

In Chapter 2 we presented various state of the art methods of time-scaling targeted to speech and music audio signals. The current chapter studies the outcome of adapting and applying these approaches to the recordings of live sports events introduced in Chapter 4. We focus almost exclusively on time-stretching results as the main objective of the research is to add an audio channel to slow motion videos, especially for speed factor $\alpha = 3$ which is currently the most commonly used in standard[1] slow motion videos.

As explained in Section 2.1, time-scaling methods rely on two fundamental hypotheses: the sinusoidal model and the local stationarity or pseudo-stationarity. The first condition is almost never met in the recordings at our disposal which contain mostly noise, even for regions where speech or harmonic sounds such as whistles are present. As for the second hypothesis, many signals studied hereafter present some form of pseudo-stationarity, but not necessarily on the same scale as usually encountered in time-scaling applications. For instance the background noise created by the crowd in a football stadium is mostly a locally stationary colored noise with slow amplitude variations. However, that local stationarity cannot be perceived on a classical time span of a few milliseconds. Instead it is observed within frames whose duration is longer by at least one order of magnitude.

As we explain in Sections 5.1 and 5.3 these differences with speech and music signals render many methods inadequate for time-scaling of sports recordings, particularly the time-domain and model-based ones, even when tuning their

[1] As opposed to more recent Hyper/Ultra/Extreme/Super slow motion videos with speed factor going as high as $\alpha = 12$ or even more.

parameters accordingly. However, we describe in Section 5.2 interesting results that have been obtained when working in the frequency domain with phase vocoders or white noise spectral modification. These methods are significantly improved in Section 5.4 when combined with transient processing. Finally, we present in Section 5.5 some promising experiments with a sound texture synthesis method tweaked to mimic audio time-stretching. Note that subjective perceptual evaluations are presented in Section 6.6.2 of the next chapter along with the results of these tests for the new method detailed in chapter 6.

5.1 Time-domain

The SOLA and WSOLA methods described respectively in Sections 2.2.3 and 2.2.4 both rely on duplication of blocks of samples to create a time-stretching effect. In the case of harmonic signals, this repetition reproduces an existing periodic pattern. However, in the case of noise signals, it creates a local periodicity that is not present in the original signal. This periodicity is perceived as an annoying buzz by the listeners. Besides, time-domain methods generally cause *metallic* distortion, even when applied to clean speech signals. These major defects do not allow the methods to be used in front of public audiences who expect at least standard quality audio from their television set.

Only a few parameters can be modified in SOLA: the length of each frame, the synthesis hopsize and the allowed shift around the theoretical overlap position in the output. As for WSOLA, the latter is replaced by the allowed shift around a theoretical position in the input. During the experiments several combination of these parameters have been tested to no avail. Notably, the length of the frames has been increased from the more common 10-20 ms up to an unusual 341.33 ms (16384 samples at 48 kHz) to take into account the fact that the stationarity in the noisy signal is maintained over longer periods than speech and music. Different synthesis hopsizes of $1/4$, $1/8$ and $1/16$ of the frame length have also been tried. Note that in the case of SOLA, this must be understood as the theoretical hopsize around which the frame is shifted, since the actual synthesis hopsize is variable.

Given the poor results obtained during the preliminary tests on the various sports available, research to adapt existing time-domain time-scaling algorithms has been abandoned early on. However, note that the novel mixed approach detailed in Chapter 6 is also, to some extent, a time-domain one.

5.2 Frequency-domain

Two approaches in the frequency domain are tested in the following with positive results in both cases. The first one is a phase vocoder with frame generation or, in other words, an inversion of a *short-time Fourier transform* (STFT) with its phases adapted to the given speed ratio α, as described in Section 2.3.1. The second method is also based on the inversion of an STFT but with random phases.

5.2.1 Phase Vocoder

As a reminder, the phase vocoder based on frame generation [44, 45] computes the STFT of the input signal, creates as many new intermediate synthesis frames as needed following Equation 2.13 and adapts the phases of all the frames according to Equation 2.15. It then proceeds to synthesize a time-stretched version of the original sound using the inverse STFT presented in Section 1.2.3. We prefer this vocoder implementation over the most common "frame shifting" because it is less computationally intensive for time-stretching than the latter. Besides, as explained in Section 2.3.1, it can stay at a playback time position for an infinite duration, literally pausing in the signal while still playing it. In any case, note that informal listening tests involving both methods have been performed regularly during this research to make sure that there is no distortion specific to the chosen approach in the results.

In order to reduce the amount of iterations needed to test the many different parametric possibilities for the phase vocoder, we used only powers of two for the frame length L. Moreover, the hopsize R_s is always set at $L/4$, although some minor tests not reported below have been conducted with $R_s = L/8$ without noticeably improving the results.

Frame Length

An immediate observation is that, with standard short-term analysis frame lengths in the order of 10 to 40 ms or more exactly 512, 1024 and 2048 audio samples at 48 kHz, there is a lot of distortion present in the extended signal, even for relatively small values of $1 < \alpha \leq 1.5$. The artifacts sound as if

fast-changing harmonics were added randomly across the frequency content of the input signal [80]. Listeners describe it as reminiscent of the noise of a little waterfall. It seems to match Boll's description in [81] and is referred to as *musical noise* in [82] and later publications. Although musical noise is a typical artifact of spectral subtraction used for speech denoising, which does not correspond to our use case, we borrow this term nonetheless in the following to designate the similar distortion we observe in time-stretching of sports recordings and noisy environment in general.

For background noises such as the crowd during a football game, another observation is that for two of these short-term analysis frames, taken close in time, their spectral contents can be different enough that, for instance, if they are used to filter white noise, they produce two sounds that are perceptually different. On the contrary, if we consider two neighboring long-term analysis frames, whose length is in the range of a few hundreds milliseconds, their spectral content is very similar and filtered white noises sound perceptually identical. Over even longer span of time, it becomes difficult to obtain relatively homogen background signals as multiple events arise, such as footstep, ball impacts, referee whistles, etc.

Consequently, we consider that the background signals of sports recordings are pseudo-stationary on a long-term time-scale, long-term being understood as a few hundreds milliseconds, as opposed to the classical short-term analysis frame length used in speech and music processing. Therefore, we progressively lengthen the analysis frames of the phase vocoder and, as a matter of fact, musical noise steadily decreases as the frame length increases, until it becomes inaudible. Depending on the sport considered and the type of sound within this sport, the frame length above which the distortion cannot be heard varies from 4096 to 16384 samples (about 85 to 341 ms). Quiet sports such as baseball or cricket can be processed reasonably well with a frame length of 4096 to 8192 samples whereas the noisiest sports like football or rugby often require a frame length of at least 16384 samples. It is not clear whether the distortion observed is directly linked to the level of noise or to the distribution of the frequency content for a given sport or even to another unknown factor.

Results and Discussions

Such long frames have a strong smoothing effect on the quick variations in the spectrum of the signal. Typically, for football, most details and small or quick variations of the spectrum, such as ball impacts, drums or whistle onsets for example, are lost or averaged, smeared across the spectrum, as the example of Figure 5.1 shows. Only the long-term parts of the signal are kept more or less intact[2]. For instance the background noise of the crowd cheers and their relatively slow variations of amplitude are well preserved, as Figure 5.2 illustrates. Likewise horns continuously playing the same frequencies in the background and stable parts of referee whistles resemble their original version although some more reverberation is present. Speech parts are more uneven in their results. Indeed, the sounds are merged into each other making some parts unintelligible whereas long and stable screams, for instance the players or referees in the tennis match, pass through the time-stretching mostly unimpacted except once more for an additional reverberation[3].

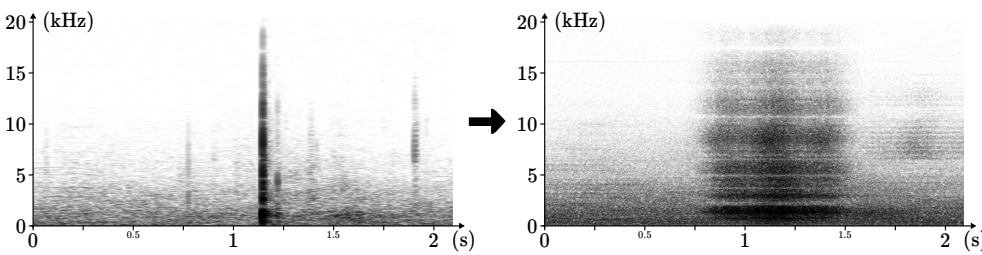

Figure 5.1. *Left: spectrogram of a ball hit by a baseball bat; right: spectrogram of the same sound time-stretched three times using a phase vocoder and a frame length of 16384 samples. The time-stretched transient is spread across several hundreds of milliseconds whereas the original one occupies about ten milliseconds. Example from file 613B.*

A potential drawback of the method is that the length of the frame creates important delays when working in realtime. Indeed, if the operator wants to start a slow motion effect directly during the live event at time t_0 then

[2] Intact as in α times slower.

[3] Note that, in most sports, the original input audio already presents some reverberation, but it is accentuated by the time-stretching.

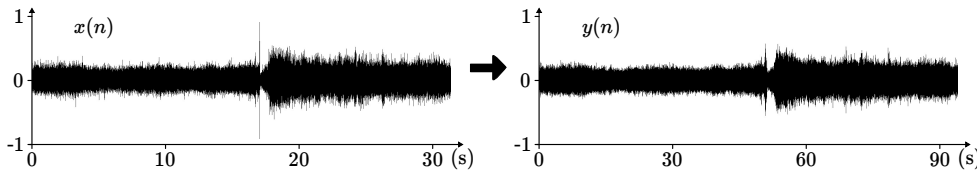

Figure 5.2. *Left: waveform of an original football recording; right: its time-stretched version. The overall shape of the amplitude over time is preserved on the long-term (on the scale of a second, or more), although slightly smoothed.*

the frame of sound that corresponds should contain the sample in the range $[t_0 - L/2, \ldots, t_0 + L/2 + H - 1]$ to be able to initialize the phase vocoder. However, audio samples $[t_0 + 1, \ldots, t_0 + L/2 + H - 1]$ are not yet available at time t_0. Therefore, the algorithm must wait for these samples before it can generate sound. When L is relatively small (a few milliseconds), the delay between audio and image is generally not noticeable on television[4], but for $L = 16384$ samples, the delay would be easily noticeable and unacceptable.

However, most of the time a slow motion video is created moments after the action is over. The operator selects a part of the video recording, creates a clip and then the director decides whether to broadcast it. In such a configuration the problem does not exist since all the samples are already recorded and thus available for the algorithm to work with. Besides, in the occasional case of "on-the-fly" slow motion, since the audio is played at a slower rate than it is recorded the problem appears only at the beginning of the slow motion sequence. For instance after one second of slow motion at a third of normal speed, only about 333 ms of sound have been used and 666 ms are still available (roughly speaking), enough to fill the 16384-sample (341 ms) audio frames completely. Therefore, a solution to this issue is to start with a small value of L, hence an imperceptible initial delay, and increase it progressively as more and more samples are available. Normally, since this has to be done for less than a second, the distortion should not be perceived. However, special care must be taken when modifying the frame size during a phase vocoder operations to avoid any discontinuities in the time envelope or the phases.

[4] Which sometimes presents other delays of its own.

Another possible delay related to the increased frame length is the slow reactivity of the algorithm. Indeed, a new frame is computed every H samples, with H the hopsize. Each synthesis frame is computed according to the position of the slow motion playback relatively to the input signal. If the playback speed factor α is constant, the frames synthesized correspond to the position in the input signal. But if the value of α changes over time, the adaptation of the time position in the audio can only happen every H samples. Therefore, in a worst case scenario, if the change occurs just after a frame is synthesized, the resynchronization between the image and the sound will happen H samples later, at the next frame. In our most common configuration with $L = 16384$ and $H = L/4$, this comes down to a delay of about 85 ms, or two video frames at 25fps. If the change happens during a stable region of background noise, it will not be perceived, but if it happens simultaneously with an action causing an acoustic transient (e.g. an impact with a ball), it may become noticeable. This problem can be solved by reducing the hopsize H, but it increases the computational cost of the algorithm proportionally which may not be acceptable for a realtime system.

Finally, as we will see in Section 5.4.5, large frames are also impractical to use for transient processing. As a matter of fact transients less than 16384 samples apart, for instance, are a common thing in many sports recordings but they need to be processed separately. This implies using smaller frame lengths, smaller than the minimal possible time span between two distinct transients and, therefore, re-introducing musical noise.

Phase Locking

Using any of the vertical phase locking methods described in Section 2.3.2 does not affect the acoustic quality of the output. This is an unsurprising and expected result since these methods have been developed specifically for signals following the sinusoidal model from Section 2.1. For instance, a peak present in the spectral amplitude of a recording does not necessarily correspond to a partial. Besides, for the most advanced approaches, tracking of sinusoidal component trajectories is necessary, which is not a trivial problem in noisy environments such as the ones present in the database of Chapter 4.

Transient Processing

For standard audio signals, adding transient detection and processing greatly improves the results for the phase vocoder. In the same way we expect it to improve the results for sports recordings. The process handling the transients and the results obtained are identical to those of the inverse STFT with random phases explained in Section 5.2.2 and, therefore, they are presented together in Section 5.4.5.

5.2.2 Random Phase

Since the major part of the signal is composed of background noise, we attempted a different approach to the phase generation than the vocoded phases of Equation 2.15. Indeed, we divide a Gaussian white noise into overlapping frames, each with the same length L and hopsize H as the analysis frames of the input signal, and compute the spectral phases of each frame. These phases are then used as the phases of the synthesis frames overlap-added into the output signal by an inverse short-time Fourier transform, as shown in Figure 5.3. As for the amplitude spectrums, they are computed through interpolation using Equation 2.13, as is done in the phase vocoder of Section 5.2.1.

Frame Length

As with the phase vocoder the level of distortion is not acceptable for standard frame sizes of 512 to 2048 samples. However, the level of distortion is lower and sounds slightly different from the "musical noise" of the phase vocoder. With this approach it sounds *buzzier* and metallic although it is not that far from the distortion observed using the phase vocoder. We use the term "musical noise" later, notably in Sections 6.4.1 and 6.4.3, to designate this distortion as well. Once more increasing the length of the frames reduces the distortion until is vanishes for frame lengths above 4096 to 16384 samples, depending on the recording that is processed.

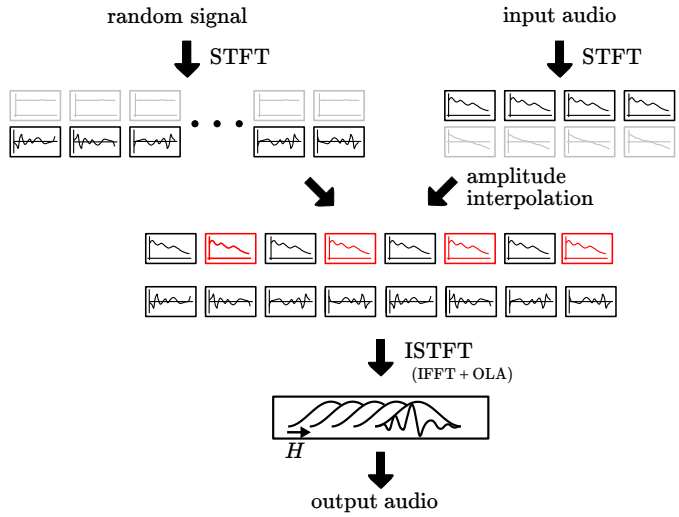

Figure 5.3. *A random signal is used to generate as many phases as necessary for the inverse short-time Fourier transform while the amplitudes are interpolated (in red) from the input signal. In the above example, $\alpha = 2$ as there are twice as many frames in the output as in the input signal.*

Results and Discussions

Although the phases are computed from a Gaussian white noise, with large frames the resulting audio sounds similar to the input. Even whistles and speech are barely degraded, and these few artifacts are acceptable because almost inaudible in the context of noisy sports. All observations and reasonings that have been made for the phase vocoder in Section 5.2.1 are applicable to this method as well, including artifacts such as transient smearing.

Note that in order to reduce the computational cost of generating random numbers and computing their FFTs, we pre-generate a few seconds of random signal and compute and store their spectral phases beforehand. Then, during the process, the algorithm uses these phases one after the other and when it has used all of them, it loops back to the first phases. We tested this approach with as little as one second of random signal without the loop being noticeable. With smaller length we start perceiving a cyclic pattern in the output audio.

5.3 Model-based

Different source-filter approaches have been tested, in all of them we use a white noise as the source signal, only the filter used changes. We tested standard *linear prediction* (LP) as well as *mel-log spectrum approximation* (MLSA) filtering [83] using time-varying filters obtained by analyzing the input audio frame by frame over time with various standard values from the literature as parameters. Filtering modifies the spectral envelope of the white noise so that it matches the spectral envelope of the input signal, only varying at a slower rate corresponding to the required time-stretching factor α. To reproduce the frame by frame variations of the filter without creating sudden changes, an interpolation is made sample after sample from the coefficients of one filter to those of the next one.

The results obtained are approximate at best and most often present many distortions. They obviously lack harmonics since the source is white noise but "fake" harmonics are created when interpolating from one filter or spectral envelope to the next between successive frames. They also lack most of the transients from the original recordings but this could be addressed with appropriate transient detection and processing. Increasing the length of the frame significantly reduces harmonic distortions but, as is always the case, smears the fast variations of the signal and makes the boundaries between otherwise separate events blurry.

We also increased the order of the filters so that they capture more of the content of the input sounds. For instance an order of about 200 for an LP filter is enough to model the presence of harmonics (referee whistle, speech, ...) for $F_s = 48$ kHz, but such a high order can make the filter close to unstable, enough to cause important resonances, and, as such, unusable for all practical purpose.

As it is the case for time-domain approaches the results are not good enough to pursue in that direction. However, linear prediction filtering is one of the possibilities that we use to fill the gap between shifted grains in the new method presented in Chapter 6.

5.4 Transient Detection and Time-Scaling

In this section, we test three detection functions presented in Section 2.6.2, namely energy, spectral flux and multi-band spectral flux. Each function is tested on different categories of sports: quiet (baseball, cricket), intermediate (tennis) and noisy (football). The files used are the ones selected for each sport as described in the relevant subsections of Section 4.1. In the three approaches the signal is divided into frames of 1024 samples with a hopsize of 256 samples or, in other words, an overlap of 75 % (i.e. 768 samples). Each frame is windowed by a Hann weighting function as defined in Equation 1.4. These parameters have been selected empirically, in the range of values usually found in state of the art publications, as a compromise between the precision of the measure and its computational cost. Indeed, although it is equivalent to a resampling of the detection functions by a factor 256, basic comparison experiments, with and without anti-aliasing filter, showed that the aliasing is all but nonexistent as each detection function occupies a very small bandwidth. Therefore, aliasing in the resulting downsampled detection functions is extremely limited and acceptable without the need for a low-pass filtering. With a larger frame shift of 512 samples, CPU usage is halved but aliasing becomes noticeable.

5.4.1 Energy

The energy of each frame of length N, windowed by a Hann function $w(n)$, is computed according to Equation 1.5 to obtain a measure $E(n)$ of the evolution of energy over time. For a digital signal $x(n)$ of length L, with an analysis hopsize H, this measure can be derived from Equation 1.6 as

$$E(m) = \sum_{n=0}^{N-1} |x(mH - \frac{N}{2} + n)w(n)|^2 \quad \text{for } m = 0, \ldots, \lfloor \frac{L - N/2}{H} \rfloor \tag{5.1}$$

The four graphs in Figure 5.4 illustrate the results obtained when applying this estimation to various sports with the chosen parameters. One can see that transients present in quiet sports such as baseball are outstandingly visible and their detection poses no particular problem. As far as tennis is concerned, the main transients are clearly identified. However, some peaks appear where no

transient is actually audible and "minor" transients such as ball rebounds or echoes have a lesser amplitude which can make them hardly detectable even though they are perfectly audible in the audio recordings. Noisier regions corresponding to referees speaking or audience applauding present many peaks, some of which cannot be paired with actual transients in the signal.[5] Lastly, football and rugby recordings, the noisiest type, highlight either no identifiable peaks or only the most significant ones. Therefore, most, if not all, events cannot be detected in this kind of audio content.

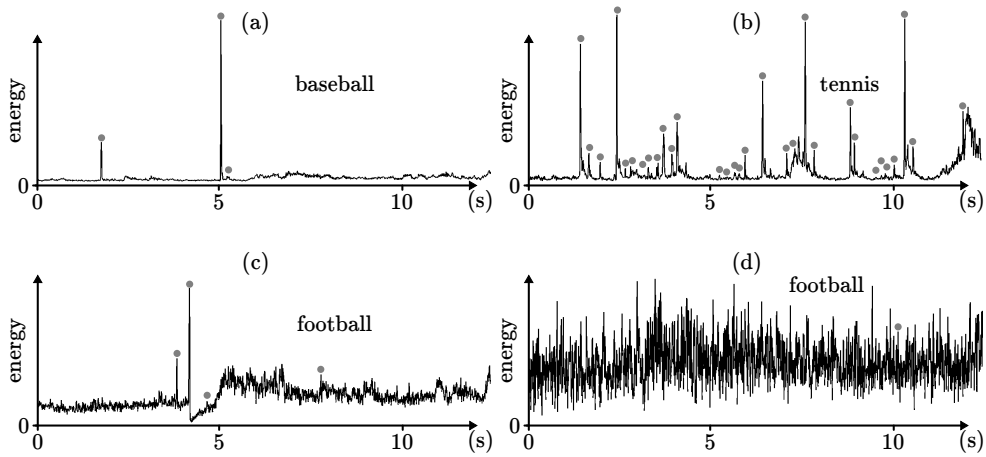

Figure 5.4. *Energy examples that are representative of the various types of sports encountered in the database. Each gray dot represents a transient. We can see that football is the noisiest sport, with cases (graph* d*) where no information about transients can be deduced whereas baseball (a) is much quieter and could use the energy as an impact detector. Tennis (b) is in the middle between these two extremes as it features large peaks for each ball hit, but some of the footsteps and rebounds are missing.*

The measure of energy is extremely sensitive to the amount of environmental noise. When noise occupies enough frequency bands with a level equivalent to the other sounds, it is not possible to make a difference between them. However, energy seems to be an appropriate tool for transient detection in sports similar to baseball for which the multi-band spectral flux presented in Section 5.4.3 would probably be an overkill.

[5] Although the definition of transient is highly subjective as explained in Section 2.6.1.

5.4.2 Spectral Flux

Using spectral flux, as defined in Equation 2.24 and explained in Section 2.6.2, on the same test set of sports recordings as in the previous section, we obtain the results presented in Figure 5.5.

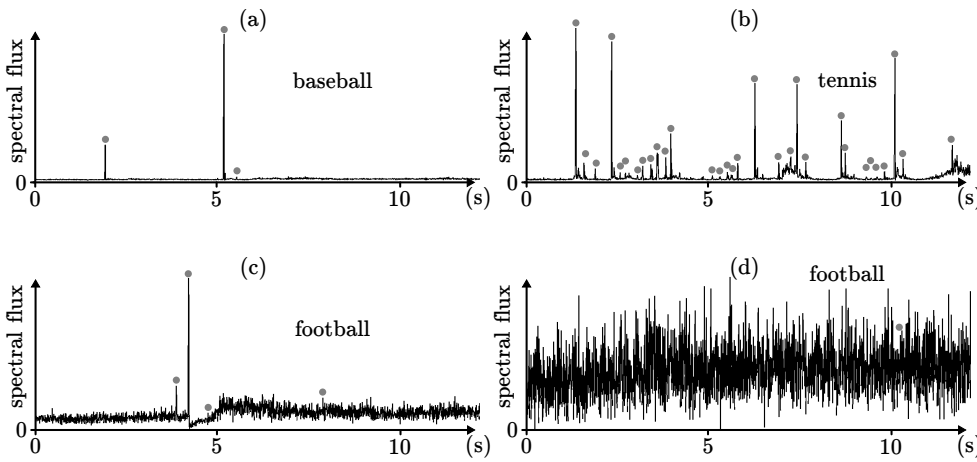

Figure 5.5. *Spectral flux for various sports from the database. Each gray dot represents a transient. Compared to the energy evolution of Figure 5.4, there is a denoising effect that highlights the transient compared to the background noise. However, the noisiest football recording (graph* d*) still cannot be searched for transients. Besides, noisier region of tennis recordings (b) have too much noise energy to allow a detection of transient.*

We observe that the peaks matching transient events are sharper and relatively larger than when using energy, compared to the spectral flux of the noisy parts of the signal which are lowered. For tennis recordings, peaks corresponding to false detection of transients in the speech regions (referees) are less pronounced than their energy counterparts. In the case of football, depending on the level of noise, some transients can be detected, but in the noisiest condition (which are commonplace during a football broadcast) most transients are still hidden by the crowd noise.

5.4.3 Multi-Band Spectral Flux

The third detection function used to detect transients in sports recordings is the multi-dimensional spectral flux defined in Equation 2.28 in which the spectrum of the signal is divided into bands spaced according to mel-scale and a spectral flux is computed for each band. For a sampling frequency of 48 kHz, we set the number of bands to $B = 42$.

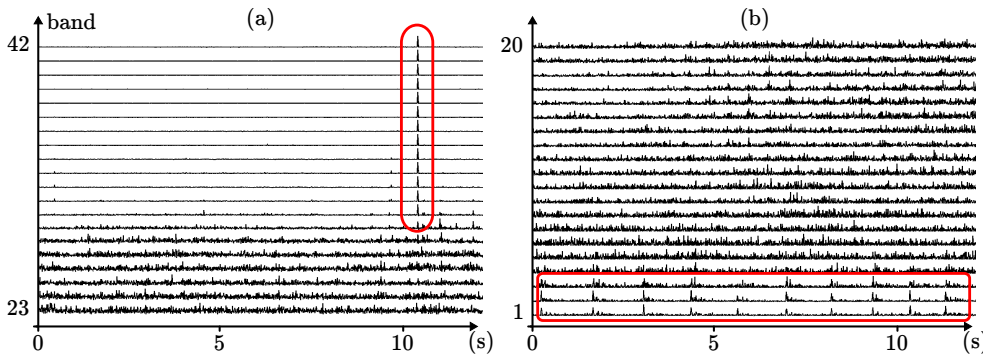

Figure 5.6. *Multi-band spectral flux. Each graph features a different subset of bands, to highlight the interesting part of the detection functions. Both excerpts come from the same signal (625F) but not the same time slot. Figure (a) corresponds to the bottom-right football example in Figures 5.4 and 5.5. It features a shoot in a ball visible only in the highest frequencies whereas Figure (b) shows regularly-spaced peaks representing drum beats in the crowd in the bands corresponding to low frequencies.*

The results for quiet sports such as baseball and cricket are not really interesting as they do not highlight any previously undetected transients. For these signals it makes more sense to use energy or spectral flux-based solutions in order to reduce the computational cost of the transient detection. However, the peaks are even more marked than with the other approaches and thus easier to locate. Therefore, the choice of a detection function depends on a compromise between the accuracy needed and the higher computational cost of a multi-band approach.

Conversely, for noisier signals, the multi-band approach highlights many events invisible in the previous detection functions and the choice of this method over

the mono-dimensional ones is necessary. This applies especially to football signals where audible transients that are not detectable with the preceding approaches are clearly visible in different regions and bands of this measure. Figure 5.6 (a) shows the detection function for the bottom right example (d) of football from Figures 5.4 and 5.5. Each band of the spectral flux is normalized on the figure but only for illustration purpose. It is not normalized during the actual process of transient detection, in order to preserve the relative importance of each band with regard to the others. In Figure (a), we observe a peak, coming from a shoot, visible across several bands corresponding to higher frequencies. Drum attacks can be seen in the lowest frequency bands of (b), apart from the noisy content of the crowd which lays in the intermediate and low frequencies.

For peak detection as explained in the next section, various approaches exist. On the one hand, we can consider all the bands as possibly containing transients, which means that the detection process must be robust to noise and not detect transients where there is actually only fast variation of noise that human ears do not perceive as transient. On the other hand, one could process only some frequency bands that are known to contain significant transients. For instance, in football the peaks corresponding to drums are in the lowest frequency bands and those for the ball shots are located in the highest bands. The middle frequency bands could be ignored in the processing. However, the peaks for the applause claps are positioned in these bands and would be discarded as well, so the choice, if any, would depend on which transients the user (director, sound engineer, ...) wants to preserve when creating a slow motion video.

5.4.4 Peak Detection

The large variety of signals encountered in sports recordings makes it impossible to use a constant threshold on the detection function. Such a threshold would need to be modified for every sports event, and most likely "on-the-fly" during a given event, depending on the evolution of the noise and environmental conditions. In other words, only an adaptive threshold seems suitable to the problem of transient detection within recordings of sports events.

Peak detection is divided into two parts. The first one finds all the significant peaks in a detection function, and the second one compares each peak to a threshold computed as a function of the neighboring values of the detection function. In the case of multi-dimensional functions, peaks are detected and compared to the threshold separately in each band. The information obtained is then recombined in a third complementary step.

Significant Peak

We define a sample $E(n)$ of a detection function as a *significant peak*, and thus as a *transient candidate*, if its value is larger than that of its $2N$ closest neighboring samples $\{E(n-N), \ldots, E(n+N)\}$. N is usually set to approximately take into account the masking effect of transients on each other. Since each sample $E(n)$ represents H samples, the farthest samples $E(n \pm N)$ are NH samples away from $E(n)$. For a value of $H = 256$, we set the value in the range $8 \leq N \leq 16$ which corresponds to 2048 to 4096 samples or about 43 to 85 ms at 48 kHz, reasonably close to the estimated range of duration of the temporal masking effect of a transient [84]. In practice this means that if two transients are less than NH samples apart, the smallest of the two is ignored by the algorithm.

Adaptive Threshold

When a peak at sample n is selected as a transient candidate, it is considered as a transient if its value $E(n)$ is higher than an adaptive threshold $T(n)$. Various approaches for the threshold have been tested during the research and are presented in Equations 5.2 to 5.4. They are based on statistics extracted from a set of values $e(n)$ of the detection function surrounding the candidate peak, with $e(n) = \{E(n-N), \ldots, E(n-2), E(n+2), \ldots, E(n+N)\}$.

$$T(n) = \lambda \mu_e(n) \tag{5.2}$$

$$T(n) = \mu_e(n) + \lambda \sigma_e(n) \tag{5.3}$$

$$T(n) = \lambda \frac{\min_e(n) + \max_e(n)}{2} \tag{5.4}$$

where λ is a "sensitivity" tuning parameter and $\mu_e(n)$, $\sigma_e(n)$, $\min_e(n)$ and $\max_e(n)$ are respectively the mean, standard deviation, minimum and maximum values of frame $e(n)$. λ can be adjusted so that the detection algorithm is more or less sensitive. Comparing the different thresholds, we found experimentally that $T(n)$ from Equation 5.3 with $10 \leq \lambda \leq 20$ is giving the most acceptable and consistent results across all sports. A large value of λ (e.g. $\lambda > 40$) means that less transients are detected and possibly that some corresponding to visually significant events in the video are missed, whereas too small a value (e.g. $\lambda < 5$) can turn every peak from $E(n)$ into a transient. Figure 5.7 shows examples with different values of λ.

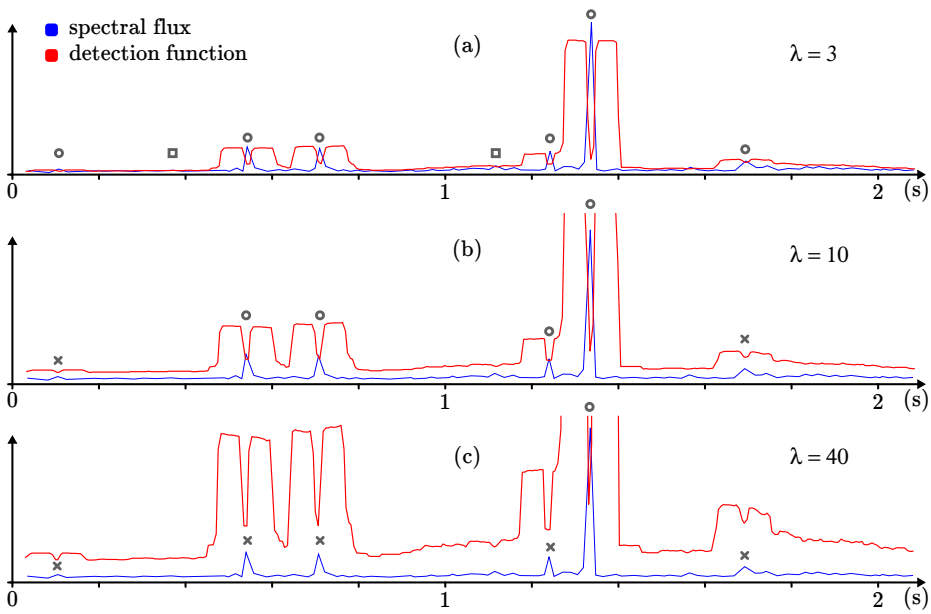

Figure 5.7. *Adaptive threshold (red) applied to spectral flux (blue) of a cricket example (130A), for various values of λ. The file contains six transients (footsteps, ball hit and rebound, etc.). In Figure (a), they are all detected (gray circles) as well as two false detections (gray squares), whereas in Figure (b) four are detected and two are missed (gray crosses). In Figure (c), only one is detected, corresponding to the ball-bat impact which is the loudest and most visually significant transient.*

Although it could be optimized for each recording, this parameter needs much less modification and adaptation than a constant threshold and can be used across many recordings of different sports with relatively little tweaking. Nevertheless we consider it a liability as it reduces the capacity of a time-stretching algorithm to automatically handle several types of sports in many different recording conditions. This is one of the reasons that led us to develop a new method, presented in Chapter 6, which does not rely on an explicit transient detection step.

Multi-Band Adaptive Threshold

In the case of the multi-band spectral flux, after each band has been processed as explained in the previous section, for each sample n, the amplitude of the peaks detected in each band are summed. Therefore, from B signals $E_b(n)$, each compared with adaptive thresholds $T_b(n)$, this creates a one-dimensional signal $E(n)$ whose values are either zero, if no peaks were found at instant n in any of the bands, or the sum of the amplitude of the peaks detected.

A second pass of peak picking is made on this signal which implicitly "merges" the peaks that are too close to be considered separately by human ears but that were located at slightly different samples in the B frequency bands. Note that no thresholding is strictly necessary in this second step since all the peaks present have already been identified as transient in the first step. However, in a post-processing step, we can decide to discard some peaks, for instance if a peak is detected in only one band across the B bands (taking into account the peaks that are merged in the last step), it could be considered as a false detection and ignored in the time-scaling process. Unless its amplitude is outstandingly high compared to those of the other bands.

5.4.5 Processing

Once transients have been located using the method described in Section 5.4.4, they need to be processed separately from the stable part of the signal. In the following we assume that these stable regions are processed using a phase vocoder with a frame length of 16384 samples and a hopsize of 4096. As we will see in this section, the unusually large frames that we use impair the

standard processes that would otherwise manage the transients properly. Two approaches described in Section 2.6.4 are tested, each with its own drawbacks explained hereafter.

Transient Removal

In this method, schematized in Figure 2.11, each transient is removed from the input signal $x(n)$ and its samples are replaced by filtered white noise. The spectral envelope of the noise is computed as an average of the spectral amplitude of the frames directly before and after the transient. In our implementation these two frames have a length $N = 4096$ and are centered at $p \pm {}^{3N}/_{4}$, with p the position of a transient and center of a N-sample frame $\{x(p - {}^{N}/_{2}), \ldots, x(p + {}^{N}/_{2} - 1)\}$. The samples $\{x(p - {}^{N}/_{2}), \ldots, x(p + {}^{N}/_{2} - 1)\}$ of the input signal are multiplied by $1 - w(n)$ where $w(n)$ is the Hann window defined in Equation 1.4. Then a N-sample frame of filtered noise windowed by $w(n)$ is overlap-added into the input signal at the same positions to create a transientless signal $\hat{x}(n)$. Eventually, an energy normalization ensures that the insertion of the filtered noise does not create amplitude variations in $\hat{x}(n)$.

In the second part of the algorithm, $\hat{x}(n)$ is time-scaled with a speed ratio α to obtain a signal $\hat{y}(n)$. Then the windowed frame containing the transient $\{x(p - {}^{N}/_{2})w(0), \ldots, x(p + {}^{N}/_{2} - 1)w(N - 1)\}$ is overlap-added to the samples $\{\hat{y}(\alpha p - {}^{N}/_{2}), \ldots, \hat{y}(\alpha p + {}^{N}/_{2} - 1)\}$ multiplied by the function $1 - w(n)$ and thus inserting the original transients into $\hat{y}(n)$ to obtain the final output signal $y(n)$, a time-scaled version of $x(n)$ with intact transients.

However, this method has a major drawback illustrated in Figure 2.11. When $\hat{x}(n)$ is time-scaled, samples located before and after the transient are used simultaneously to create the output samples. Therefore, sounds that ought to be heard only before the transient are still audible after it and vice versa. Obviously this mix also occurs in the stable parts of the signal but it is barely noticeable for sports recordings and causes little to no trouble, especially when the stable part is made of slowly varying crowd noise as in football. Conversely it can cause important artifacts around transient events.

An outstanding example we encountered during the tests, is the case of a tennis service, where the impact sound is preceded by the whooshing sound of the racket slicing the air and followed by a wobbling sound. When this method

is applied to this signal, the two sounds get mixed up and the resulting signal is perceived as different from the sound expected from a service. Another example is the beginning of a referee whistling where a pre-echo of the whistle can be heard before the referee actually starts blowing.

We argue that this artifact is perceptible because of the oversized frames used within the phase vocoder. Indeed, with standard frame sizes, as used in speech processing, it would happen only for a few milliseconds before and after a transient and, as such, it would generally be perceptually hidden by the transient sound. In our configuration though, the duration over which the problem occurs is in the order of a hundred milliseconds because of the frame length used. As such, it is too large to benefit from this masking effect.

Another problem arises when the algorithm does not process a transient because it is too close to another one and thus perceptually masked. If the transient that is ignored lies within one of the two frames used to replace the transient that is processed, it means that the algorithm replaces a transient with another one, albeit less significant, which will be smeared by the phase vocoder step. Finally a related problem is the case when a sound contains many transients over a short span of time. Replacing them with filtered noise creates a section of signal with high density of artificial sounds which generates poor quality audio once processed.

However, this approach gives better results in general than the one described in the following section. Moreover, the perceptual tests presented in Section 6.6.2, as well as informal discussions with viewers, show that the various artifacts inherent to this method remains often acceptable, as long as the most visually significant transients are correctly reproduced.

Local Speed Variations

In this method, illustrated in Figure 2.12, the regions surrounding a transient are extended more than necessary (i.e. with a stretching factor larger than the actual α) to compensate for the fact that the transient region is not extended (i.e. it uses a stretching factor equal to 1). In theory, this should give an output signal as close as possible to an ideal time-scaled signal. In practice though, this makes it impossible to process transients less than $L+H$ samples apart, with L the frame size of the phase vocoder and H the hopsize. In our

configuration, this means for transients less than 20,480 samples or 427 ms apart. Even if the hopsize is changed to a much smaller fraction of L in this particular case, the condition is still that no transient should be closer than about 350 ms. By using a value of 8192 samples for L, we can possibly reduce this minimum spacing to about 175 ms, but it is still too large compared to the minimum perceptible spacing between two transients.

In all cases, transients closer than these values is a condition frequently met with sports audio signals. Of course, a solution to this problem is to temporarily reduce L to a value smaller than the time lapse between two successive transients. But then musical noise distortions reappear and although it is for relatively small durations, they can become clearly noticeable and annoying.

However, an advantage of this method compared to the previous one is that the sounds positioned after a transient in the original signal do not affect the sound before a transient in the time-stretched signal. Consequently in every portion of sports recordings where transients are more than $L + H$ samples apart, the acoustic quality of the transients is correctly preserved.

5.5 Sound Textures

In [71], Parker et al. propose a method for sound texture synthesis that we modified to generate an audio texture that mimics the effect of audio time-scaling. Their method consists first in dividing an audio signal into frames and then in reorganizing and duplicating these frames almost randomly for as long as needed. The sequence of frames obtained is not completely random as Parker et al. add a continuity constraint over frame selection. In the output audio, the m^{th} frame f_m is selected so as to maximize similarity between the overlapping part of f_{m-1} and f_m. Note that the frame that follows f_{m-1} in the original audio is discarded from selection to avoid reproducing the input audio. Besides, an integer value called "selection cost" is added to each selected frame so that it is not selected again immediately. This cost is then decremented at every iteration until it is back to zero and the frame can be used again.

We modify several aspects of this algorithm to adapt it to our need of time-scaling and to fix some of its limitations. A first change is that we pick the

selection cost of each frame randomly in a range of values $[C_1, \ldots, C_2]$, as opposed to setting it to a constant value in [71]. This makes it impossible for the selection process to be stuck in a loop over a few frames, as it can happen in the original method, instead of parsing the whole set of frames. A second change is that we use cross-correlation to detect the best matching frame whereas Parker et al. use a least-square similarity measure. The last and most fundamental change we make to obtain a time-scaling effect on a signal $x(n)$ is that instead of selecting frames in a set created from the whole $x(n)$, our method goes forward in $x(n)$ and modifies the set of frames over time so that it contains only the frames surrounding the current position in $x(n)$ over a relatively short span of time (in the order of a few hundred milliseconds). This way we can generate local textures of arbitrary length for each region of the input signal and create a new signal that imitates the time structure of the input signal but with a different duration.

The method has been tested on football and tennis recordings with frames selected over a span of time of one second. Informal listening tests showed that it correctly synthesizes the stable part of signals such as the crowd or the referee whistle but creates artifacts around transients and quick variations. For instance when a referee starts whistling, the whistle attack can be followed by one of the frame of noise located around the attack in the input signal. Besides, the main artifact of the method is that events are not actually slowed down; instead, they last longer. For instance during the applause period in tennis file 610C, for a time-scaling by a factor $\alpha > 1$, people applaud α times longer instead of α times slower. The nuance is that there are more claps in the pseudo time-stretched version than in the original signal. This is of course an unwanted effect since the extra claps do not correspond to actual events in the original recordings. However, the good results in stable parts of the signal, such as background crowd noises, show that sound texture synthesis is a potentially interesting solution for creation of artificial ambient noises of arbitrary length for sports recordings.

5.6 Conclusions

Among the methods introduced in Chapter 2 and tested in the present chapter, the most promising results have been obtained using spectral-domain approaches with large analysis frame size, combined with transient detection and processing. The results obtained in stable sound sections or sections with few transients have a high quality, most probably adequate for broadcast television. Baseball recordings, for instance, can generally be time-stretched with a phase vocoder combined with transient detection through spectral flux and local speed variations, without the need for further processing, as the results in Table 6.4 (Section 6.6.2) show. Besides, these tests and informal discussions from Section 6.6.2 have shown that viewers are generally satisfied with the results as long as transients linked with the most visually significant events are handled correctly.

For most sports however, occasional or recurrent artifacts are caused on the one hand by the difficulty to detect all transients in noisy recordings and across many varieties of signals, and on the other hand by oversized frames. This led us to the conclusion that another approach is needed to process such transients properly. We think that the direction taken in Chapter 6, which implicitly manages transients without actually detecting them, is more adapted to the kind of signals recorded during sports events.

Chapter 6

Slowdio

In this chapter we detail an original approach based on relevant hypotheses about the audio content in sports recordings. We coined the term *slowdio* to designate our method and its outcome. It is the contraction of *slow* and *audio*, in reference to *slowmo* which is used to casually refer to slow motion videos.

In Chapter 2 we presented state of the art algorithms for time-scaling and applied them in Chapter 5 to a new type of data, the sports recordings presented in Chapter 4, in order to assess their capacity to adapt to audio signals that do not fit with their underlying hypotheses. Both the phase vocoder and white noise filtering show interesting results when used with oversized frames, for stable or slowly evolving parts of the example signals. When combined with transient detection and processing, the resulting sounds present an acceptable quality as long as transients are relatively sparse. Of course, this also implies that all transients and their position are correctly detected. This is rarely the case in real-life conditions, as sports recordings often exhibit rapid successions of transients in extremely noisy environments, making their detection and processing unreliable and prone to distortion.

The method presented in the following sections assumes that most of the signal is made of noise and contains a significant amount of transient events that have to be preserved, as opposed to the sinusoidal model of Section 2.1. Besides, contrary to most state of the art time-scaling methods, we consider that frames with variable size, which we call *grains* from now on, are more adapted in order to handle properly the diversity of sounds, and especially transients, that can be produced and recorded during a sports event. Finally, we reckon that the large overlap between analysis frames used in all the methods from

the state of the art is not adapted to our situation. The overlap works with speech-like signals because of their pseudo-periodicity. Indeed, two successive and overlapping frames contain at least 75% of samples in common, albeit time shifted, and the remaining samples are usually similar in both frames due to the pseudo-periodicity. The time-stretching methods make use of this redundancy and of the pseudo-periodicity to stretch the signal into a similar sound of longer duration. We have explained in Chapter 5 that using the same approach for sports signals can create cyclic patterns that are absent from the original signal and distort the time-stretched sound. Moreover, it is one of the causes of transient duplication, an artifact described in Section 2.6.1, since a transient, like any other part of the signal, is present across several overlapping frames[1] and, if not detected and processed properly, is reproduced at various positions in the output signal. Therefore, in the following, we choose to extract, from the sports recordings, audio grains that do not overlap and that are used only once in the output.

In the next section, we give a general overview of the principles behind this time-scaling method. The process can be decomposed in three parts, which we detail in Sections 6.2, 6.3 and 6.4 with various possible approaches in each case. Section 6.5 introduces the notion of a variable speed factor α, as is generally used in practice for slow motion videos of sports events. Finally, Section 6.6 discusses the different results obtained, such as a C++ implementation and formal and informal tests.

A publication entitled *"Audio Time-Scaling for Slow Motion Sports Videos"* has been accepted for publication at the *16th Digital Audio Effects Conference (DAFx-13)* [85]. It describes the general principle of the method as explained in Section 6.1 and its implementation using cepstral self cross-synthesis from Section 6.4.3, as well as the results of the MOS and CMOS tests as detailed in Section 6.6.2. Moreover, on April 7, 2011, we applied for a European patent entitled *Time-Stretching of an Audio Signal* [86]. This was later extended, on April 6, 2012, for a worldwide protection [87]. The two applications have been published in October 2012, respectively on the 10$^{\text{th}}$ and the 11$^{\text{th}}$. At the time of writing their status is pending (A1). The scope of the patent covers the various approaches described in this chapter.

[1] For instance it is contained in four successive frames if the overlap is 75%.

6.1 Overview

Section 2.7.2 introduced a method for time-scaling presented by Picard et al. in [73, 75]. This method decomposes a database of sounds into grains and computes so-called *correlation patterns* between each grain and the recordings in the database. A time-stretching method is then proposed by Picard et al. that shifts the grain of a recording to new positions, corresponding to the desired speed factor α and inserts other grains from the database in between, selecting those that maximize the correlation pattern with the sound that is processed. This method cannot be used in realtime because it needs to compute the correlation patterns of every grain of the database with every file. This is very time consuming and, as such, cannot be implemented within a realtime application. Besides, it supposes that a set of recordings is already available as opposed to our situation where we start "from scratch" for each new slow motion excerpt. Adding the necessity to keep a database of recordings in memory is too strong a constraint for our goal of embedding audio in the realtime slow motion videos as generated by EVS systems.

We propose a new method that also decomposes the input file into non-overlapping grains that are re-spaced according to a speed factor α, and then the empty spaces left between two grains are filled with content generated "on-the-fly". Furthermore, each grain has a certain "freedom of movement" around its theoretical position in order to minimize artifacts.[2] This approach can be summarized as a three-step *"split-shift-fill"* algorithm illustrated in Figures 6.1 and 6.2 and whose steps are detailed in the following sections.

Note that with this approach, the entirety of the input signal is reproduced exactly in the output signal, grain by grain, only with empty spaces to be filled in between. For instance a time scaling by a factor $\alpha = 1.25$ has 80% of its samples coming directly from the original recording. Even for $\alpha = 3$, the audio of the input sound still occupies a third of the output audio. This is similar to standard time-domain algorithms. However, contrary to these, there is neither grain duplication nor change of hopsize, hence no transient duplication. This means that all the transients, attacks and transitions are preserved in the time-scaled version, avoiding an unreliable transient detection in noisy environments. Besides, as we show in Section 6.4, the inter-grain gaps

[2] As will be explained in Section 6.3.

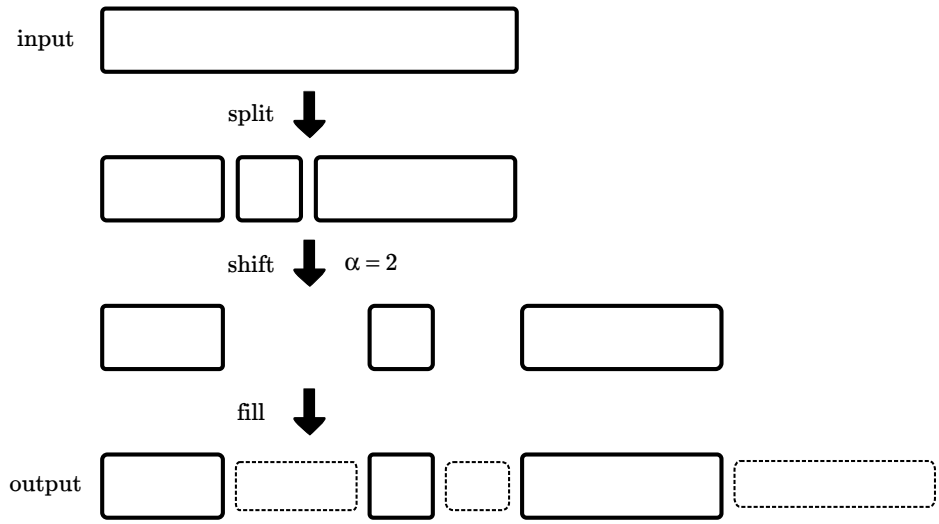

Figure 6.1. *An input signal is split into non-overlapping variable-length frames, the so-called grains. Empty spaces are inserted between grains to match the speed factor α (equal to 2 in this example). Finally, synthetic content (dashed blocks) is generated to fill the gaps in the output signal.*

are filled with content that, for most scenarios, is not generated using a time-domain approach. Instead, the best results are obtained by filtering white noise either with a linear prediction filter or by modification of the spectral amplitude. In both cases, the filtering is based on analysis of the audio samples around the segmentation point between the two grains surrounding the gap.

Figures 6.1 and 6.2 illustrate the method from the signal and process point of views, respectively. Figure 6.1 is an over-simplification since the actual process presents significant improvements that are detailed hereafter. Figure 6.2 summarizes schematically the ordering of each step of the method for one iteration of the algorithm (i.e. the life cycle of a grain). First a grain is extracted as described in Section 6.2, then we test whether it can be concatenated directly to the previous grain or it has to be shifted as Section 6.3 explains. If it has to be shifted, content is generated beforehand, using one of the approaches

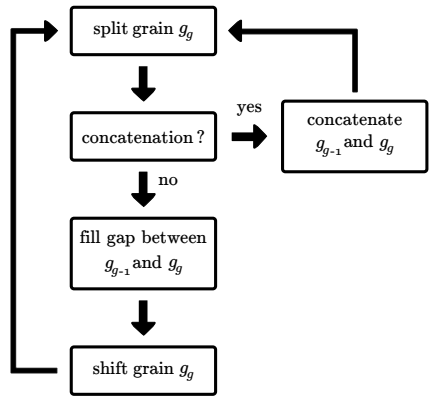

Figure 6.2. *Life cycle of the g^{th} grain g_g during the process. After a grain g_g is extracted from the input, it is either concatenated to the previous grain, g_{g-1}, or shifted. In case it is shifted, synthetic samples are first generated and inserted to fill the gap after the previous grain and then the new grain is positioned and inserted.*

detailed in 6.4 and then the grain can finally be shifted in time and inserted at an optimal position in the output signal.

6.2 Grain Extraction

The first part of the algorithm consists in dividing the input audio into grains whose boundaries are located at stable or non-transient samples. In other words, the segmentation samples should never belong to a transient and a grain should always contain either one or several complete transients or no transient at all. Picard et al. [73] propose to place segmentation points at minimums of the spectral flux of the audio signal. However, when applying this approach to databases of sports recordings, we observed several segmentation errors[3] and found that segmentation based on the energy followed by a search for either the closest zero-crossing or a close minimum in the audio waveform is more adapted to our needs, as explained in more details below.

[3] For some subjective definition of *error*, obviously. Something we deem a mistake may or may not be considered as such in another context.

6.2.1 Segmentation

The first step in the segmentation process of a signal $x(n)$ is to compute a *transientness* function as explained in Sections 2.6.2 and 5.4.4. In the present case however, our goal is to detect non-transient parts of the signal, in order to place the grain boundaries at location where they do not split a transient into two successive grains. As we show later on, spectral flux is sometimes unreliable and we obtain more correct segmentations using the energy measure $E(n)$ as defined in Equation 5.1.

In a second step, the g$^{\text{th}}$ grain g_g is defined by two parameters, the position t_g of its first sample[4] and its length L_g, in number of samples. The length L_g of a grain is simply equal to the distance between its starting point and the starting point of the next grain

$$L_g = t_{g+1} - t_g \tag{6.1}$$

The computation of t_g is illustrated in Figure 6.3 and complete details of the process can be found in Equations 6.2 to 6.7.

$$B_{min} = \lfloor \frac{t_{g-1} + L_{min}}{H} \rfloor \tag{6.2}$$

$$B_{max} = \lfloor \frac{t_{g-1} + L_{max}}{H} \rfloor \tag{6.3}$$

$$E_s(m) = \{E(B_{min}), \ldots, E(B_{max})\} \tag{6.4}$$

$$p = (\arg\min(E_s(m)) + B_{min})\ H \tag{6.5}$$

$$x_s(m) = \{x(p - \frac{N}{2}), \ldots, x(p + \frac{N}{2} - 1)\} \tag{6.6}$$

$$t_g = \arg\min(|x_s(m)|) + p - \frac{N}{2} \tag{6.7}$$

with the first grain starting at the first sample of the signal and thus $t_0 = 0$. L_{min} and L_{max} respectively correspond to the lower and upper limit of a grain length, in number of samples. N is the frame length and H the frame hopsize,

[4] Other meaningful options that could be considered are the middle sample of each grain or the one corresponding to the maximum of energy inside the grain (i.e. the most likely to be a transient that we want to keep synchronized with the video). However, these would add a slight overhead to the method without changing the practical results much.

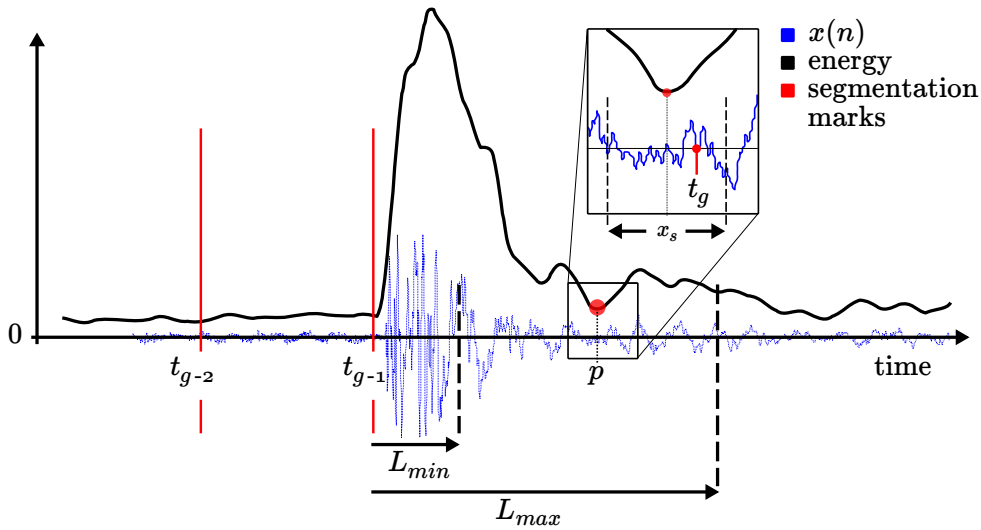

Figure 6.3. *The energy of an audio signal $x(n)$ is computed and its local minimums are used to place temporary segmentation points p which are then refined into actual segmentation points t_g. Example from tennis file 610C.*

as used in Equation 5.1 to compute the energy function $E(n)$. p as computed in Equation 6.5 is the position of the minimum of $E(n)$ located between L_{min} and L_{max} samples after t_{g-1}, the beginning of grain g_{g-1}. This position is then refined in Equation 6.7 by looking for the value of $x(n)$ closest to zero in a N-sample neighborhood around $x(p)$. This last step can be replaced with no important impact by a search for the closest zero-crossing. Note that we do not look for the minimum *peak* in $E_s(m)$[5] because the absolute minimum is more likely to not be part of a transient than any other local minimum. Note also that since the limit of $x_s(m)$ are set $\pm N/2$ samples around p, it means that the final value of L_g is actually in the range $[L_{min} - N/2, \ldots, L_{max} + N/2]$.

6.2.2 Parameters

We empirically set the computation parameters for $E(n)$ to $N = 256$ for the frame length and $H = 4$ for the hopsize, which for a sampling rate $F_s = 48$

[5] As opposed to what is done in Equation 3.9 of Section 3.1.1, for instance.

kHz corresponds respectively to 5.3 ms and 83.3 μs. The value of H is of little importance, as the cost of computing $E(n)$ is relatively low, but it could be increased to further reduce that cost, if need be. On the other hand, the low value of N, compared to usual lengths found in the literature, is important since, for longer frame sizes, each transient would influence $E(n)$ over a larger span. This, in turn, would blur the location of minimal values that are essential to the segmentation method. To put it differently, reducing N makes it more likely that some of the frames used to compute $E(n)$ are devoid of any transient and, as such, are neat and valid segmentation points. However, too small a value for N produces a noisy $E(n)$ (closer and closer to $x(n)^2$ as N decreases) that is also inappropriate for segmentation.

The other constraint applied during segmentation is related to the duration of grains. In the literature on granular synthesis, grains typically have a variable length in a range somewhere between 1 and 100 ms. We arbitrarily fixed the minimum duration of a grain in our implementation to 10 ms and decided for a maximum length of 40 ms which corresponds to the duration of a video frame at 25 fps and, based on prior study of the test database, seems a reasonable maximum duration for transients. In number of samples, this translate into $L_{min} = 480$ and $L_{max} = 1920$ samples. Note that, as explained in the previous section, these constraints are applied loosely and in some specific cases the grain size can be slightly out of these limits, within the range $[L_{min} - {}^{N}/_{2}, \ldots, L_{max} + {}^{N}/_{2}]$. Therefore, it may happen that a grain is shorter than 10 ms or longer than 40 ms by 2.67 ms (i.e. $\frac{N}{2F_s}$). Nevertheless, this has no consequences on the remainder of the process which can handle any grain size.

6.2.3 Discussions

In a first implementation of the segmentation algorithm, we used spectral flux instead of energy, following the idea of Picard et al. [73]. However, we encountered occasional errors where a transient with duration significantly longer than N was split between two grains because it had few spectral variations locally and thus exhibited a minimum in its spectral flux as illustrated in Figure 6.4. This happened for instance with some tennis services or some impacts with a football ball. Such transient signals have significant energy all along, in-

cluding within their "stable" sections. Therefore, the search for a minimum of energy is more adapted to their segmentation while avoiding unwanted splits.

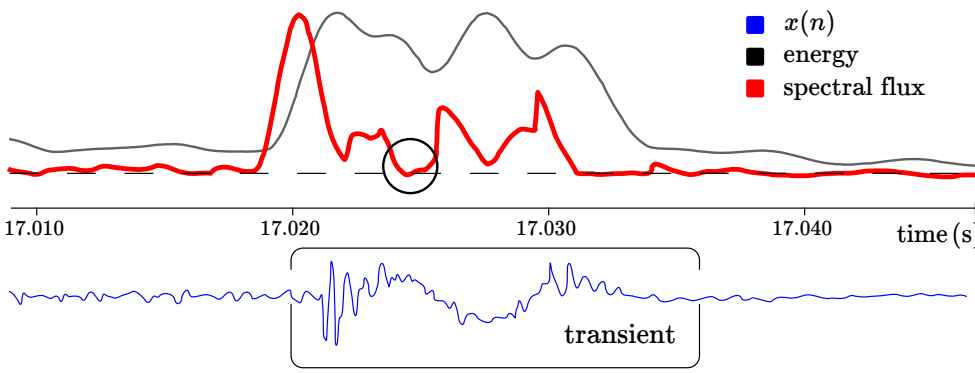

Figure 6.4. *Spectral flux as defined in Equation 2.24 presents a local minimum (circled) during an important transient thus potentially triggering a false detection of "non-transientness" while the energy level remains high enough to avoid the error. Example from football 548D around* $t = 17.03$ *s.*

Another approach we tested is to use a spectral flux measure to locate a first position and then to search for an energy minimum close to that position and eventually for zero-crossing or waveform minimum nearby that energy minimum. This approach seemed unnecessarily complicated and, in practice, results are close to those obtained using only energy. However, if segmentation problems appear in the future due to the use of energy alone, this may be a good alternative and we will reconsider our decision of not using it.

Finally, contrary to state of the art methods, this approach does not require an explicit detection of transient events and thus eliminates the need for thresholding which is usually specific to the recording processed and has to be constantly re-adapted. Indeed, as long as transients are not split between successive grains, all transients that may appear in the input signal $x(n)$ are implicitly processed in the next stages of the algorithm.

6.3 Grain Shifting

In the second part of the method, grains are shifted from their initial position t_g in the input $x(n)$ to a new position u_g in the output signal $y(n)$, in order to create the time-scaling effect. Theoretically, for a constant[6] speed factor α, the new position ought to be

$$u_g = u_{g-1} + \alpha L_{g-1} \tag{6.8}$$

However, although this places the grains exactly where they belong, it adds a strong constraint whether α is larger or smaller than 1. As a matter of fact in case $\alpha > 1$, the methods detailed in Section 6.4 have to generate content that fits perfectly into the gap created between two grains. Conversely, in the case of $\alpha < 1$, it forces an overlap-add between g_{g-1} and g_g at a position that is likely to be suboptimal, in the same way the time-domain method OLA does as explained in Section 2.2.3. Our experimental observations show that such an approach is both prone to artifacts and unnecessarily restrictive. To overcome this issue, we apply two improvements.

On the one hand, as presented in Section 6.3.1, we use the principle from SOLA [24] detailed in Section 2.2.3 and that we already applied to PVSOLA [50] in Chapter 3. We allow grains to be shifted around position u_g, by δ_g samples with $|\delta_g| \leq \Delta$, in order to minimize discontinuities when grains are overlap-added with the content already in $y(n)$. For reasonable values of Δ, the change in synchronization between the sound and the image in a slow motion video cannot be perceived while the acoustic quality, especially in region with harmonic content[7], is significantly increased.

On the other hand, as introduced in Figure 6.2, we show in Section 6.3.2 that, provided some conditions are met, the insertion method of Section 6.3.1 can be advantageously replaced by a concatenation between two successive grains. This reduces the computational cost of the algorithm and increases its acoustic quality by removing as many possible discontinuities.

[6] Variable speed factors are discussed in Section 6.5.
[7] Referee wistles, player's voices, vuvuzelas, etc.

6.3.1 Shift

The computation of the insertion positions of each grain resembles that of the *reset frames* in PVSOLA, using a cross-correlation measure $\chi(n)$, defined in Equation 6.11, to choose a value of δ_g that minimizes discontinuities. $\chi(n)$ is computed between $\iota(n)$, the first h samples of grain g_g, and $o(n)$, the samples of $y(n)$ located $\pm\Delta$ samples around sample u_g.

$$\iota(n) = \{x(t_g), \ldots, x(t_g + h - 1)\} = \{g_g(0), \ldots, g_g(h - 1)\} \tag{6.9}$$

$$o(n) = \{y(u_g - \Delta), \ldots, y(u_g + \Delta - 1)\} \tag{6.10}$$

$$\chi(m) = o(n) \star \iota(n) \tag{6.11}$$

where h is the number of samples at the beginning of g_g that are going to be overlap-added with the output signal $y(n)$. Note that the computation of $\chi(n)$ supposes that the output samples $\{y(u_g - \Delta), \ldots, y(u_g + \Delta)\}$ exist. In other words, if, as is often the case for $\alpha > 1$, some of these samples are located after g_{g-1}, that is after sample $y(u_{g-1} + \delta_{g-1} + L_{g-1})$, the gap between $y(u_{g-1} + \delta_{g-1} + L_{g-1})$ and $y(u_g + \Delta)$ has been previously filled with generated content obtained from one of the methods described in Section 6.4. Once cross-correlation has been computed, a search for an absolute maximum peak is made on a subset $\chi_s(n)$ corresponding to a shift $|\delta_g| \leq \Delta$ of g_g around u_g

$$\delta_0 = 0 \tag{6.12}$$

$$\varepsilon = 2\Delta \tag{6.13}$$

$$\chi_s(n) = \{\chi(\varepsilon), \ldots, \chi(\varepsilon + 2\Delta - 1)\} \tag{6.14}$$

$$p = \arg\ \text{maxpeak}(|\chi_s(n)|) \tag{6.15}$$

$$\delta_g = p - \Delta \tag{6.16}$$

OverLap-Add

Grain g_g is inserted into $y(n)$ at position $u_g + \delta_g$ through overlap-add (OLA). The first h samples of g_g are windowed by $w(n)$, the left half of a $2h$-sample Hann window, as defined in Equation 1.4, and the last h samples of the output signal are windowed by the complementary window $1 - w(n)$, which also happens to be the right half of the $2h$-sample Hann window. Figure 6.5 and Equations 6.17 to 6.19 present the overlap-add procedure. Note that in case the

maximum peak of correlation is negative (i.e. the samples are anti-correlated), the values of the grain samples are sign-inverted before OLA.

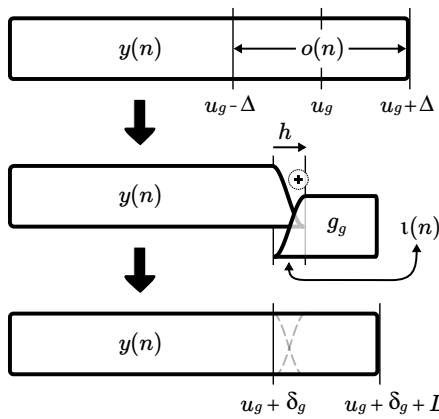

Figure 6.5. *The g^{th} grain and the current $y(n)$ are windowed by half of a $2h$-sample Hann window and its complement, respectively. Samples of $y(n)$ located after $u_g+\delta_g+h$ are discarded. Then g_g is overlap-added to $y(n)$, starting at position $u_g + \delta_g$, to obtain the next version of $y(n)$.*

$$m = u_g + \delta_g + n \tag{6.17}$$

$$\varsigma_g = \operatorname{sgn}(\chi_s(p)) \tag{6.18}$$

$$y(m) = \begin{cases} y(m)(1 - w(n)) + \varsigma_g \; g_g(n)w(n) & \text{if } n = 0, \ldots, h-1 \\ \varsigma_g \; g_g(n) & \text{if } n = h, \ldots, L_g - 1 \end{cases} \tag{6.19}$$

where ς_g is equal to either 1 or -1 depending on the sign of $\chi_s(p)$. The second case of Equation 6.19 is equivalent to a windowing by a rectangular window. This combination of a Hann window for the first h samples and a rectangular window for the remainder corresponds to the first half of a *Tukey* window.

6.3.2 Concatenation

If u_g is less than Δ samples away from the end of the previous grain, g_{g-1}, inserting g_g into $y(n)$ becomes a simple concatenation of the two grains. In-

deed, if Equation 6.20 is verified, the optimal location for g_g is to put it right after g_{g-1} in $y(n)$, as illustrated in Figure 6.6, just like they follow each other in the input signal $x(n)$, ensuring perfect continuity in the output signal.

$$u_g - \Delta \leq u_{g-1} + \delta_{g-1} + L_{g-1} \leq u_g + \Delta \tag{6.20}$$

with $u_{g-1}+\delta_{g-1}$ the index at which the first sample of g_{g-1} has been previously inserted in $y(n)$. This double condition checks that $u_{g-1} + \delta_{g-1} + L_{g-1}$, the first sample after g_{g-1} in $y(n)$, is within $u_g \pm \Delta$, the range of available positions for insertion of g_g. Note that only the first inequation is useful when $\alpha > 1$ and, likewise, the second is a necessary and sufficient condition when $\alpha < 1$.

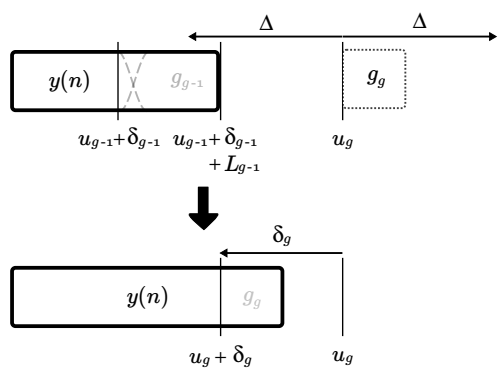

Figure 6.6. *Theoretically, grain g_g should be inserted at position u_g. However, a shift Δ is allowed and, in some cases, it is possible to append g_g directly to g_{g-1} in the output signal $y(n)$. This reduces the computational cost.*

Taking Equation 6.8 into account, Equation 6.20 can be rewritten independently from grain g_g as

$$(\alpha - 1)L_{g-1} - \Delta \leq \delta_{g-1} \leq (\alpha - 1)L_{g-1} + \Delta \tag{6.21}$$

which, in turn, can be finally simplified into

$$|\delta_{g-1} - (\alpha - 1)L_{g-1}| \leq \Delta \tag{6.22}$$

The condition depends only on the shift applied to g_{g-1}, on its length L_g, and on the speed factor α. It is independent of any parameter from g_g. However,

as discussed in Section 6.5, this is only valid for a constant speed factor or a value of α that is updated once per grain. If the condition is respected, the grain is added directly into the output as per Equations 6.23 to 6.25. Note, however, that it is multiplied by ς_{g-1}, as computed for g_{g-1} during the previous iteration of the algorithm. Indeed, grain g_{g-1} has actually been multiplied by ς_{g-1} before insertion and thus may have been inverted. There would be a discontinuity in case $\varsigma_{g-1} = -1$ and g_g is inserted without being inverted.

$$m = u_{g-1} + \delta_{g-1} + L_{g-1} + n \tag{6.23}$$

$$\varsigma_g = \varsigma_{g-1} \tag{6.24}$$

$$y(m) = \varsigma_g \; g_g(n) \qquad \text{for } n = 0, \ldots, L_g - 1 \tag{6.25}$$

In the concatenation case, as opposed to Equations 6.12 to 6.16, δ_g is computed using Equation 6.26, where it is the only unknown. Equation 6.26 can be simplified into Equation 6.27 by subtracting the terms of Equation 6.8. The value of δ_g that is obtained has no immediate use, but it is needed in the next iteration of the algorithm, when grain g_{g+1} is added to $y(n)$.

$$u_g + \delta_g = u_{g-1} + \delta_{g-1} + L_{g-1} \tag{6.26}$$

$$\delta_g = \delta_{g-1} + (1 - \alpha) L_{g-1} \tag{6.27}$$

This simplification of the grain insertion algorithm by means of direct concatenation is especially useful for values of α close to one since it both reduces the computational cost and improves the acoustic quality of $y(n)$. Indeed, for such ratios, concatenation happens frequently and each time it happens the algorithm avoids the computation of cross-correlation χ from Equation 6.11 and the generation of content to fill[8] a gap, as detailed in Section 6.4. Computationally-wise, these two operations are the most expensive of the whole algorithm and the speed gain is significant, especially for values of $\alpha \leq 3$, as shown in Table 6.1.

On the other hand, each replacement of an overlap-add by a concatenation step increases the overall acoustic quality. Indeed, it reduces the number of potential discrepancies due to poor overlap-add conditions. It also suppresses

[8] Note that in the method we use in the tests, the cross-synthesis approach from Section 6.4.3, there is another cross-correlation in Equation 6.42 in order to find the best insertion location for the synthetic content. This is avoided as well when concatenating grains.

Table 6.1. *Percentage of grains that are concatenated as a function of α for a file with 966 grains. Values of α are typical of sports slow motion videos and correspond respectively to $^{24}/_{25}$, 90%, 80%, $^{2}/_{3}$, 50%, 33%, 25% and 20%.*

α	1	1.042	1.11	1.25	1.5	2	3	4	5
%	100	94.7	86.2	71.8	52.7	28.7	8.3	2.3	0.5

cases where a grain g_g could be shifted at a position located before $u_{g-1}+\delta_{g-1}$[9] or, in other words, before g_{g-1}, thus inverting some sounds in the output signal compared to the input. Besides, in the case $\alpha = 1$, the algorithm is transparent and $y(n) = x(n)$, which is not necessarily the case when only the cross-correlation-based OLA approach is used.

Table 6.1 shows the average percentages of frames that are concatenated for various typical values of α for one tennis file, identified as 610C, which is divided into 966 grains. Since the content filling inter-grain gaps is generated from filtering a random Gaussian signal, as explained in Section 6.4, the position of each grain is different every time the program is run and, consequently, the concatenations do not always occur for the same grains. Therefore, the measurements reported in Table 6.1 are made five times for each value of α and the results are averaged in the table. Note that the variance of these measurements is relatively weak.

As we can see, for values of $\alpha \leq 1.5$, more than 50 % of the grains are directly concatenated, up to about 95 % for a speed ratio of $^{25}/_{24}$. This translates into an equivalent speed up of the method as the computational cost of concatenation is negligible when compared to that of overlap-add.

6.3.3 Parameters

The two parameters of this part of the algorithm are Δ and h. In the same way that we decide the maximum length L_{min} of each grain, we choose a value of Δ that corresponds more or less to 20 ms, for a total span $[-\Delta, \ldots, \Delta]$ close to the 40 ms duration of a video frame at 25 fps, so that a shift δ_g does not perceptibly desynchronize the audio and image streams. Initially, we fixed

[9] i.e. if $\alpha L_{g-1} < \Delta$ and cross-correlation detects an optimum there.

$\Delta = 1024$ samples at $F_s = 48$ kHz, i.e. about but not exactly 20 ms, to keep a power-of-two value in order to have the fastest possible computation of χ using FFTs.[10] It is of minor importance though, as the acoustic quality of the output is not overly sensitive to that value, as long as it allows a reasonable shift. The second parameter is set to $h = {}^{N}/_{2} = 128$, with N the frame size used in Section 6.2.2.

6.4 Filling Gaps

After grain extraction, when a grain g_g has to be positioned, the concatenation condition of Equation 6.20 or one of its variants is computed. Then, as shown in Figure 6.2, if g_g cannot be concatenated, the third part of the time-scaling process is to synthesize and insert content in the output signal $y(n)$ to fill the gap between g_{g-1} and g_g. Samples need to be generated until at least sample $u_g + \Delta + h - 1$, so that $o(n)$, and thus cross-correlation $\chi(n)$, can be computed properly in Equations 6.10 and 6.11, respectively. In order to create and insert the content that fills the inter-grain empty spaces, we tested different approaches that are detailed in the following sub-sections.

6.4.1 Spectral Synthesis

Since the sports recordings in the database are mainly made of noise, we considered filtering Gaussian white noise as an option to fill the inter-grain spaces between g_{g-1} and g_g. The method from Section 5.2.2 explains how to use white noise to generate a random signal with a given spectral amplitude using the inverse STFT. To compute this spectral amplitude, we use the DFT of a Hann-windowed N-sample analysis frame $\{x(t_g - {}^{N}/_{2}), \ldots, x(t_g + {}^{N}/_{2} - 1)\}$ centered on t_g. Then we generate Z samples of *colored noise*, $z(n)$, with Z equal to

$$Z = (\alpha - 1)L_g + h + \Delta - \delta_{g-1} + \Theta \tag{6.28}$$

[10] However, with the most performing solutions for the FFT algorithm, it is possible to use almost any size while partly preserving the high speed of computation, so this decision may be modified in future implementations.

where h, Δ and δ_{g-1} correspond to the parameters already defined in Section 6.3. The actual insertion position for g_g (i.e. $u_g + \delta_g$) is still unknown, since it cannot be inserted until the gap is filled with content. However, we know its upper limit since $\delta_g < \Delta$, which explains the "$+\Delta$" in Equation 6.28. Compared to the actual number of samples needed to fill the largest possible gap between g_{g-1} and g_g, there are Θ extra samples. This is because we compute a cross-correlation measure between samples $\{y(u_{g-1} + L_{g-1} - h), \ldots, y(u_{g-1} + L_{g-1} - 1)\}$ and the first Θ samples of the synthetic content. This measure is used to find the shift θ of $z(n)$ that minimizes the discontinuities with the current last h samples from $y(n)$. Then, samples $\{z(\theta), \ldots, z(\theta + (\alpha - 1)L_g + h + \Delta - \delta_{g-1})\}$ are multiplied by -1 if the correlation value was negative in θ and eventually inserted into the output signal, starting at position $u_{g-1} + L_{g-1} - h$. Note that in the region $[u_{g-1} + L_{g-1} - h, \ldots, u_{g-1} + L_{g-1} - 1]$, the content of $z(n)$ is overlap-added to the existing samples in $y(n)$.[11] Then, starting from sample $u_{g-1} + L_{g-1}$, the samples are simply set to the values of $z(n)$.

Once colored noise has been added to $y(n)$, the next grain can be positioned and inserted as well, following the overlap-add method in Section 6.3.1. More details about the computation of the cross-correlation and the overlap-add used in the previous paragraph are given in Section 6.4.3 and Figure 6.14 as the insertion process used there is exactly the same.

Discussions

The value for Θ is set arbitrarily to 4096 samples (about 85 ms). We tested this method for various size N of the analysis frame centered in t_g. Similarly to the filtered white noise in Section 5.2.2, $N = 512$ creates strong distortions (*musical noise*). Increasing the value of N progressively reduces this distortion which becomes mostly inaudible for $N \geq 8192$. At $N = 8192$, according to several informal listening tests, the results for this method present a good quality in stable regions. All the minor events and slow transitions are correctly preserved and the background noises sound similar to the input signal, especially in football and rugby.

[11] which, incidentally, are the last h samples of g_{g-1}.

However, in sections where there is one or several significant transients, or other important variations, these interfere with the computation of the content that fills the gaps. Indeed, for $N = 8192$ the duration of the analysis frames is about 170 ms. Even a transient located 80 ms away from t_g[12] still has an influence on the content inserted between g_{g-1} and g_g, as a comparison of the spectrograms between an input signal and a slowdio shows, respectively in Figures 6.7 and 6.8. We can see that the main transient is surrounded by vertical stripes in the slowdio image. These represent the spectral amplitudes of the synthetic content that is used to fill the gaps between grains and they are clearly not in the continuity of the original spectrogram.

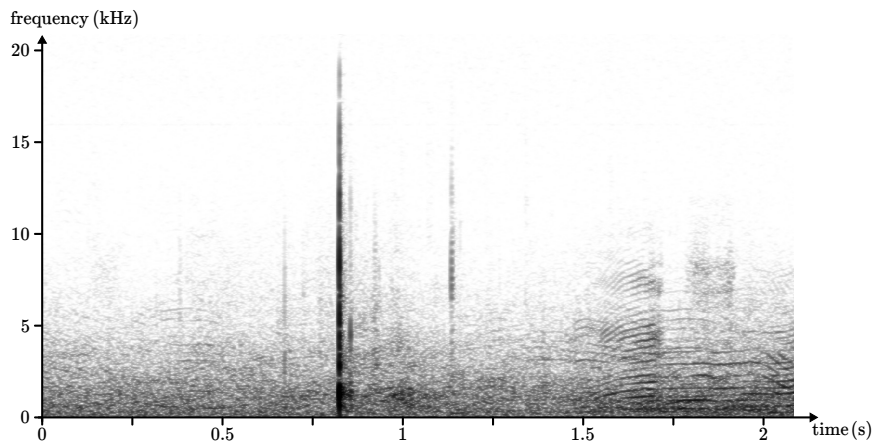

Figure 6.7. *Excerpt of a spectrogram of a baseball recording (613B, see Section 4.1.7) at the moment a ball is hit by a bat. The hit is represented by the darkest vertical line around 0.8 s.*

6.4.2 Linear Prediction Filtering

As an alternative to spectral synthesis we propose to use linear prediction filtering as defined in Section 1.5 to create the filling content with the required spectral envelope. This method has the advantage that it can ensure perfect

[12] and, as such, not belonging to g_{g-1} or g_g whose maximum length is about 40 ms.

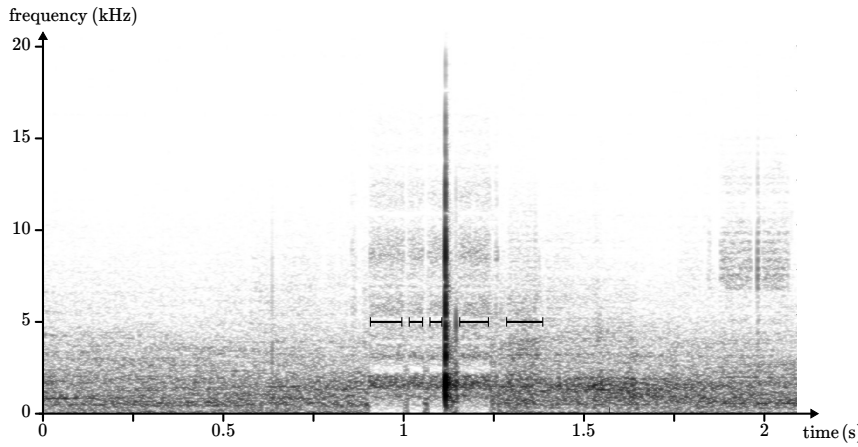

Figure 6.8. *Excerpt of a spectrogram of a time-stretched ($\alpha = 3$) version of the recording of baseball from Figure 6.7, showing synthetic content between 0.8 and 1.4 s (marked with horizontal bars). The content in between each is a grain coming directly from the original signal. Note that there is no link between the time axes of this figure and Figure 6.7.*

continuity between either grain g_{g-1} or grain g_g or both and the synthetic content. As opposed to spectral synthesis which uses overlap-add for insertion, creating the possibility for a small glitch each and every time.

One-sided

In the so-called "one-sided" approach, the content is generated in the continuity of the samples of one of the two grains, g_{g-1} or g_g. The following explanation is for a continuation of g_{g-1} as the principle for g_g is exactly the same, only applied time-reversed, as we mention at the end of the section.

The first part of the process is similar to the spectral method with the extraction of an N-sample analysis frame centered at t_g. The p linear prediction coefficients for this frame are computed and are used as the coefficients of an AR filter to filter Gaussian white noise. A simple way to obtain samples in the continuity of the last samples of g_{g-1} is to use the values of these samples

in Equation 1.35 for $y(n < 0)$, as per Equation 6.29

$$y(n) = g_{g-1}(L_g + n) \quad \text{if} \quad -p < n < 0 \tag{6.29}$$

where $y(n)$ refers here to the output of the LP filter in Equation 1.35, not the output signal of the time-scaling process. Using Equation 6.29 means that the value of $y(0)$, for instance, is computed as the sum of $bx(0)$, i.e. a random value, and $\sum_{i=1}^{p} a_i \; g_{g-1}(L_g - i)$. In other words, the filter uses the last p samples of g_{g-1} as its past outputs and continues to synthesize samples starting from that point on. This guarantees the continuity between the end of g_{g-1} and the beginning of the synthetic content. Therefore, there is no need for an overlap-add or a correlation-based adjustment of the position of the synthetic samples as it is the case in Sections 6.4.1 and 6.4.3. Then g_g can be positioned and overlap-added as explained in Section 6.3.1 and Figure 6.9.

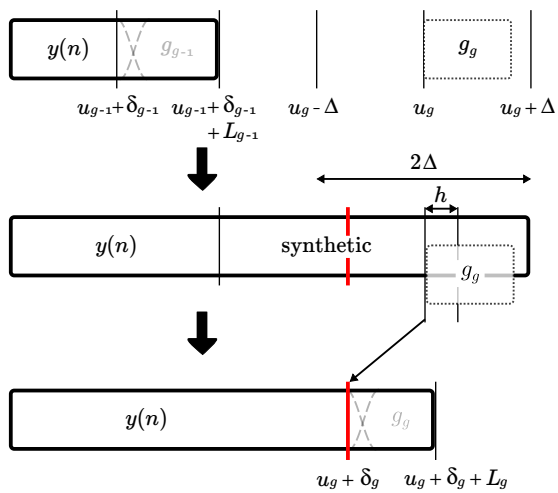

Figure 6.9. *$y(n)$ is continued by content synthesized by LP filtering of a Gaussian white noise. The optimal insertion position for g_g (red bar) is then found using a cross-correlation measure between the last 2Δ samples of the synthetic content and the first h samples of g_g.*

If this process is applied using the first p samples of g_g, time-reversed as in Equation 6.30, as the values for $y(n < 0)$ and then the output of the filter

is reversed along the time axis, we obtain a signal whose last[13] samples are continued by the first samples of g_g.

$$y(n) = g_g(-n-1) \quad \text{if} \quad -p < n < 0 \tag{6.30}$$

In this case, as illustrated in Figure 6.10, it is the whole set of samples "synthetic + g_g" that must be positioned and inserted at the end of g_{g-1} using the correlation-based method. There is no particular reason to use this second method, except in the context of the so-called "two-sided" method explained in the next section.

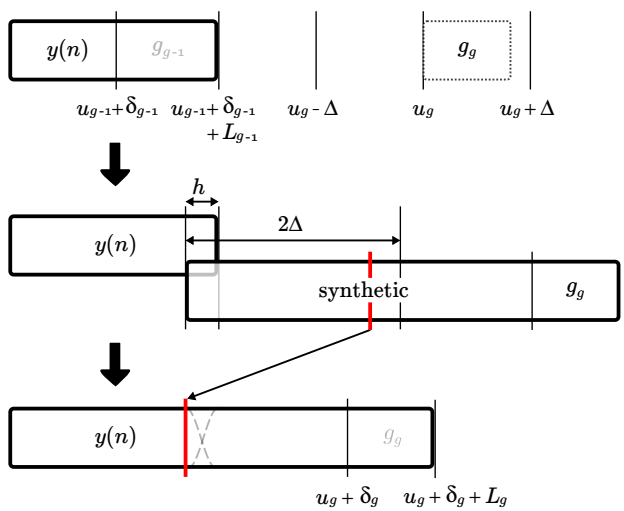

Figure 6.10. *g_g is "pre-completed" by LP filtering of a Gaussian white noise to fill the gap between its farthest possible position ($u_g + \Delta$) and $y(n)$. h extra samples are generated for the overlap-add with g_{g-1}. The grain and the synthetic content optimal insertion position (red bar) is then found using a cross-correlation measure between the first 2Δ samples of the synthetic content and the last h samples of $y(n)$.*

[13] In other words, the first output samples of the filter before time-reversal.

Two-sided

Another way to use LP filtering is to perform the two filtering steps explained hereinabove and then to overlap-add the two blocks of samples obtained as schematized in Figure 6.11. As in other parts of this work, a cross-correlation measure is used to minimize the discontinuities in the overlapping regions. This method ensures the continuity between the grains and the synthetic content on both sides of an inter-grain gap. However, it requires twice as much filtering as in the one-sided version. Besides, the potential discontinuities are not completely suppressed. Instead they are only moved from the grain boundaries into the overlapping region between the two synthetic contents. This can be advantageous when the overlapping region is much larger than h as it reduces the perception of discontinuities.

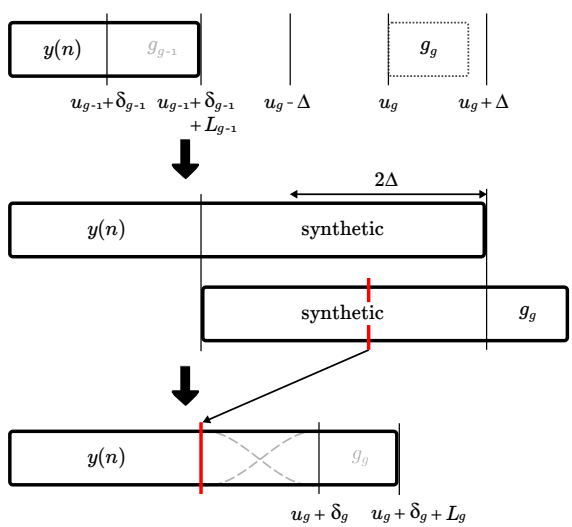

Figure 6.11. *$y(n)$ and g_g are both "continued" by content synthesized by LP filtering of a Gaussian white noise. The synthetic signals are then overlap-added at a position that minimizes discontinuities.*

Discussions

As far as the length of the analysis frame is concerned, the same observations as for the spectral synthesis can be made about this method and the length of the frame has to be at least $N = 8192$ to reduce the typical musical noise artifact. As for the order p of the LP filter, in most cases, when used in speech processing, it is chosen according to the arbitrary rule $p = F_s + 2$, where the sampling frequency F_s is in kHz. However, in the present situation, an order $p = 50$ is not enough to reproduce the fine details of the audio content and the synthetic signal is only a poor approximation of it. It results in strong audible artifacts in the time-stretched signals. The transition between grains, which are high-quality samples from the original signal, and the synthetic content, which is of a lesser quality, are clearly audible. Increasing significantly the value of p, up to 250, can reduce most of the discrepancies. As an interesting consequence of such a high value for p, the harmonic content is modeled as well while it is almost lost for LP filter with a lower order. This means that speech and whistle parts of the original signal are reproduced in a better way in the output signal. Nevertheless, some discontinuities remain, especially in the voice parts.

The use of LP filtering has several drawbacks. The first one is that the sound has a typical *metallic* characteristic which is particularly audible and annoying in the noisiest sports such as football and rugby. The second is that a high-order LP filter can more easily be subject to instabilities than a low-order one. Even though methods exist to keep it stable, the time response of such a filter can present sudden rises of energy locally. This overflows the 16 bits audio dynamic range.[14] Besides, this method has the same artifacts around transients as the spectral method since the analysis frame length is the same.

[14] Note that switching to 24 bits would not reduce the annoying acoustic artifact caused by such bursts in the energy of the synthetic signal.

Note on Direct and Backward Linear Prediction filtering

When using linear prediction, the grains are neither overlap-added nor exactly concatenated[15] with the synthetic content. Instead one could say they are "continued" by that content, as opposed to the description of Section 6.3.

6.4.3 Self Cross-Synthesis

The results from spectral synthesis and LP filtering, respectively in Sections 6.4.1 and 6.4.2, are promising, but suffer from major artifacts, mainly around transients. To tackle this issue, we propose a new approach to generate the gap-filling synthetic content, without the need to detect the transient events. We make the hypothesis that we can combine the advantages of short-term and long-term analyses through a form of cross-synthesis. Indeed, on the one hand, a short-term frame extracted around time t_g has a spectral envelope devoid of the influence of surrounding transients. Unfortunately, it causes musical noise. On the other hand, a long-term frame does not create that artifact but its spectral amplitude is distorted by nearby transients. Cross-synthesis is a process by which the spectral amplitude of a *carrier* or source signal $s(n)$ is modified into that of a *modulating* or target signal $t(n)$ [11][16]. However, in the method that we propose, the source and target signals are the same input signal $x(n)$, only the time resolution of the analysis is different, hence the name *self cross-synthesis*.

Overview

The principle of the self cross-synthesis method is illustrated in Figure 6.12. In order to fill the empty space between g_{g-1} and g_g, two analysis frames f_s and f_l are built from the input signal $x(n)$, both centered at t_g, as written in Equations 6.31 and 6.32. The lengths of the frames are respectively N_s and N_l, with $N_s << N_l$. Therefore, f_s is called the *short-term analysis frame* and

[15] Note that the concatenation between grains explained in Section 6.3.2 happens also with this method, when possible. The current section is only about cases when content has to be inserted between two grains.

[16] https://ccrma.stanford.edu/~jos/sasp/Cross_Synthesis.html

f_l the *long-term analysis frame.*

$$f_s(n) = \{x(t_g - \frac{N_s}{2}), \ldots, x(t_g + \frac{N_s}{2} - 1)\} \tag{6.31}$$

$$f_l(n) = \{x(t_g - \frac{N_l}{2}), \ldots, x(t_g + \frac{N_l}{2} - 1)\} \tag{6.32}$$

For both frames, spectral envelopes are extracted and the spectral envelope of f_l is compensated for. Therefore, it is "whitened" into a signal whose spectral amplitude is globally horizontal while still containing all its local details. Then this "whitened" signal is "re-colored" using the spectral envelope of f_s, the short-term analysis frame. Finally, the signal resulting from this self cross-synthesis is used to synthesize the content to fill the inter-grain empty space.

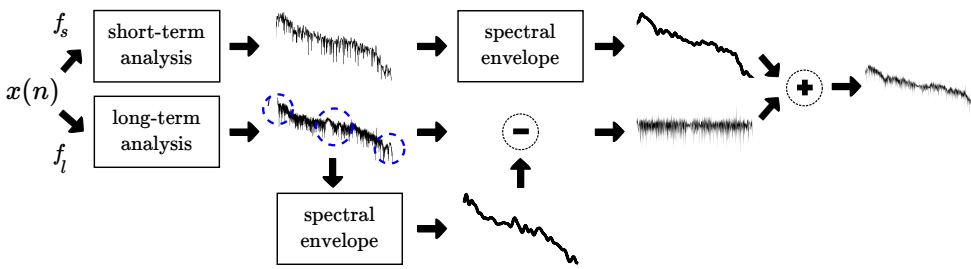

Figure 6.12. *Process of cross-synthesis of a signal with itself at different time resolutions. The contribution of the long-term spectral envelope is removed (⊖) from long-term amplitude spectrum and the contribution of the short-term spectral envelope is added back (⊕). The artifacts caused by a nearby transient in the long-term analysis (dashed blue circles) are absent from the cross-synthesized amplitude spectrum on the right.*

For each part of this process, different implementations can be used. The computation of the spectral envelopes can be achieved either through spectral or LP analysis[17]. Likewise whitening and re-coloring can be performed with LP inverse and direct filtering or with modifications of the spectrum. Finally, synthesis can also be done using either spectral or LP filtering from Sections 6.4.1 or 6.4.2. Obviously, from the performance point of view it is much

[17] Remember that the frequency response of an LP filter obtained by analyzing a frame $f(n)$ is the spectral envelope of that frame, see Section 1.5.

more efficient to use either an LP-only or a spectral-only approach instead of switching from one to the other between two steps of the process. Given the metallic artifact and the potential for quasi-instabilities of LP filtering, a spectral-only framework seems more appropriate to us. In the next section we detail a way to carry out this spectral-only self cross-synthesis process using the cepstral domain.

Cepstral cross-synthesis

Cross-synthesis in the spectral domain is usually achieved by division and multiplication of $|F_l(k)|$ with the spectral envelopes of $|F_l(k)|$ and $|F_s(k)|$, respectively [11][18]. However, when the spectral envelope are computed using cepstral liftering, it is possible to implement this directly in the cepstral domain using cepstral concatenation [88]. Both implementations are mathematically equivalent but the concatenation is less computationally intensive. Indeed, not only does it replace multiplication and division operations with a simple concatenation but it also operates only one inverse cepstral transform instead of two to convert the resulting cepstrum back into the spectral domain.

For a given signal $x(n)$, an interesting property of its real cepstrum $c_x(n)$, briefly introduced in Section 1.3.1, is that each cepstral coefficient represents a different "level" of detail of $|X(k)|$, its amplitude spectrum. More exactly, the first coefficients, the lower quefrencies, represent a gross approximation of the amplitude, as can be seen on Figure 1.10, for the computation of $y(n)$. The higher the quefrency, the finer the details. In other words, the low quefrencies are representative of the spectral envelope. Therefore, if one keeps only the first C coefficients, sets the values of $c_x(n)$ to zero for $n \geq C$ and then transforms this biased cepstrum back into the spectral domain, the resulting amplitude spectrum is an envelope of $|X(k)|$. The value of C determines the level of details, or *coarseness*, of the envelope with higher values meaning more details. This is the property that we use to compute $|F_\chi(k)|$, the spectral cross-synthesis between the two amplitude spectrums $|F_s(k)|$ and $|F_l(k)|$ of frames f_s and f_l, as illustrated in Figure 6.13.

First we compute $F_s(k)$ and $F_l(k)$, the discrete Fourier transforms (DFTs) over N_l points of $f_s(n)$ and $f_l(n)$, both weighted by Hann windows. $N_l - N_s$ zeros

[18] https://ccrma.stanford.edu/~jos/sasp/Cross_Synthesis.html

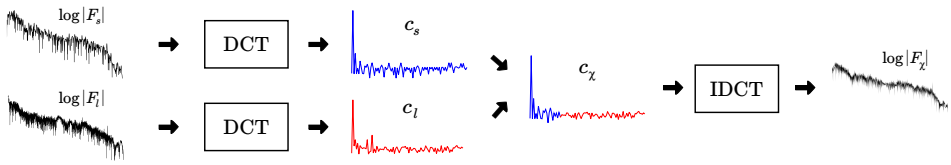

Figure 6.13. *The short-term (top) and long-term (bottom) amplitude spectrums are cross-synthesized through concatenation of the first C cepstral coefficients of the short-term real cepstrum (blue) and the $N_c - C$ coefficients of the long-term one (red).*

are appended at the end of the windowed $f_s(n)$ before computing its DFT, so that $F_s(k)$ is also N_l-sample long. Then, from $|F_s(k)|$ and $|F_l(k)|$, we compute $c_s(n)$ and $c_l(n)$, the cepstral transforms, using the discrete cosine transform (DCT) of their logarithms, as presented in Equations 6.33 and 6.34. We then proceed in Equation 6.35 to the creation of a new cepstrum by concatenation of the first C quefrencies of $c_s(n)$ and the last $N_c - C$ of $c_l(n)$. Finally, we compute the cross-synthesized amplitude spectrum $|F_\chi(k)|$ by inverting the DCT and exponentiation, as shown in Equation 6.36

$$c_s(n) = DCT(\log_{10}|F_s(k)|) \tag{6.33}$$
$$c_l(n) = DCT(\log_{10}|F_l(k)|) \tag{6.34}$$
$$c_\chi(n) = \{c_s(0), \ldots, c_s(C-1), c_l(C), \ldots, c_l(N_c-1)\} \tag{6.35}$$
$$|F_\chi(k)| = 10\hat{}(IDCT(c_\chi(n))) \tag{6.36}$$

with N_c equal to the number of cepstral coefficients. Note that the DFTs $F_s(k)$ and $F_l(k)$ have N_l frequency bins but are symmetrical since they are obtained from real-valued samples. Therefore, to reduce the computational cost of the cepstral transform, we compute only the first half of each DFT in Equations 6.33 and 6.34 and $N_c = {}^{N_l}/_2$ cepstral coefficients. Consequently, the inverse DCT in Equation 6.36 returns only a half-spectrum, but the complete amplitude spectrum $|F_\chi(k)|$ is easy to obtain as it can be assumed to be symmetrical as well. As explained in Section 1.3.1, the use of the DCT on the first half of the spectral log-amplitudes is simply an optimization to the computation of the cepstral coefficients, dividing the cost by a factor two.

For some appropriate values of C discussed later, the resulting set of amplitudes $|F_\chi(k)|$ has the spectral envelope of f_s, hopefully free from any artifact caused by the surrounding transients. It also has the fine details from f_l, which better capture details such as whistles, speech, and the long-term stationarity of the background noise of most sports such as football. Note that for $C = 0$, the transformation does nothing and is equivalent to the spectral synthesis of Section 6.4.1 with a long frame f_l. Similarly for $C = N_c - 1$, the process is equivalent to spectral synthesis, but using a very short frame f_s and, consequently, exhibiting the musical noise that goes with it.

Low-Pass Filter

In practice, we noticed that when applying this self cross-synthesis method, the spectrogram of the time-scaled signals around transients has the expected aspect but the musical noise is present again. It is lighter than with small values of N using spectral synthesis, but audible enough to be annoying. Since it does not appear when using only f_l (i.e. $C = 0$) as is done in Section 6.4.1, it is certainly due to the replacement of the first C cepstral coefficients with $\{c_s(0), \ldots, c_s(C-1)\}$ in Equation 6.35. In order to remove this last defect, we add one more step to the process which smoothes the evolution over time of the short-term cepstral coefficients, $c_s(n)$, using a first order low-pass filter.

Let us consider $c_{s,g}(n)$ the n^{th} cepstral coefficient computed from the short-term frame $f_{s,g}$ centered at instant t_g. The low-pass filter is written

$$\hat{c}_{s,0}(n) = c_{s,0}(n) \tag{6.37}$$

$$\hat{c}_{s,g}(n) = (1-\lambda)\ \hat{c}_{s,g-1}(n) + \lambda\ c_{s,g}(n) \tag{6.38}$$

and for each gap between g_{g-1} and g_g, Equation 6.35 is rewritten into 6.39.

$$c_\chi(n) = \{\hat{c}_{s,g}(0), \ldots, \hat{c}_{s,g}(C-1), c_l(C), \ldots, c_l(N_c-1)\} \tag{6.39}$$

Of course, it is only necessary to apply Equation 6.38 to the first C coefficients of c_s. Besides, it is important to note that the low-pass filtering must be used at every segmentation point t_g, including those between grains that are directly concatenated in the output signal[19]. This means that the optimization brought

[19] See Section 6.3.2 for more details about grain concatenation.

by the concatenation step becomes only a partial one. Indeed, there is still no need to compute any cross-correlation or to generate a synthetic gap-filling, but the cepstrum of f_s has to be computed and added to Equation 6.38. This is not a very expensive operation though, and the gains in terms of quality and computational cost, obtained by concatenating grains whenever possible, are preserved overall.

Note that in 2007, Breithaupt et al. published an article [89] about using cepstral smoothing for speech denoising without musical noise. This cepstral smoothing is all but identical to the low-pass filtering we apply to the short-term cepstrum c_s in Equation 6.38. The only difference is that they apply the smoothing to the coefficients above a given quefrency k_{min}, where we apply it to the ones below C. However, k_{min} is set to 4 where we use larger values of C, so all the coefficients that we smooth (and use in Equation 6.39) are low-passed as well in their implementation, but for the first four. Note that their smoothing parameter $1-\beta$, equivalent to our λ, is set to 0.2 (or $\beta = 0.8$), with some tests conducted at 0.3. This is coherent with the value $\lambda = 0.25$ that we selected[20]. Note also that for some quefrencies, corresponding to the cepstrum of the pitch excitation, they use a different value of $\beta = 0.4$.

Insertion

Once the spectral envelope $|F_\chi(k)|$ is computed, it is used in an inverse short-time Fourier transform with phases from a Gaussian white noise to synthesize the content filling the inter-grain space. The insertion method is the same as the one used in Section 6.4.1 and is illustrated in Figure 6.14. Z samples of colored noise, $z(n)$, are generated, with Z defined in Equation 6.28 already used in Section 6.4.1.

$$Z = (\alpha - 1)L_g + h + \Delta - \delta_{g-1} + \Theta \tag{6.28}$$

[20] See subsection *Discussions* hereafter.

Then we compute the cross-correlation measure χ between the first Θ samples of $z(n)$ and the last h samples of g_{g-1}.

$$\iota(n) = \{z(0), \dots, z(\Theta - 1)\} \tag{6.40}$$

$$\begin{aligned} o(n) &= \varsigma_{g-1}\ \{g_{g-1}(L_{g-1} - h), \dots, g_{g-1}(L_{g-1} - 1)\} \\ &= \{y(u_{g-1} + \delta_{g-1} + L_{g-1} - h), \dots, y(u_{g-1} + \delta_{g-1} + L_{g-1} - 1)\} \end{aligned} \tag{6.41}$$

$$\chi(m) = o(n) \star \iota(n) \tag{6.42}$$

In the same way it is done in Sections 3.1.1 and 6.3.1, the maximum peak of the absolute value of χ_s, a subset of χ, defines the position θ of the part of $z(n)$ that overlaps best with the end of g_{g-1} in the output signal.

$$\varepsilon = \Theta \tag{6.43}$$

$$\chi_s(n) = \{\chi(\varepsilon), \dots, \chi(\varepsilon + 2\Theta - 1)\} \tag{6.44}$$

$$\theta = \arg\ \text{maxpeak}(|\chi_s(n)|) \tag{6.45}$$

Finally, the selected part of $z(n)$ is multiplied by ς, the sign of $\chi_s(\theta)$, and overlap-added to the output signal at the end of grain g_{g-1}, following Equations 6.46 to 6.48.

$$m = u_{g-1} + \delta_{g-1} + L_{g-1} - h + n \tag{6.46}$$

$$\varsigma = \text{sgn}(\chi_s(\theta)) \tag{6.47}$$

$$y(m) = \begin{cases} y(m)(1 - w(n)) + \varsigma z(\theta + n)w(n) & \text{if } n = 0, \dots, h - 1 \\ \varsigma z(\theta + n) & \text{if } n = h, \dots, Z - \theta - 1 \end{cases} \tag{6.48}$$

with $w(n)$ a $2h$-sample Hann window, already introduced in Equation 6.19 and defined in Equation 1.4. Note that, contrary to what happens in the overlap-add and concatenation processes, the sign ς does not need to be propagated. Indeed, the next thing to be inserted will be grain g_g with the overlap-add method and, thereby, with its own sign ς_g, as computed in Equation 6.18.

Discussions

There are several constant parameters to take into account in this method: N_s, N_l, λ and Θ. The parameters used during formal tests are $N_s = 1024$, $N_l =$

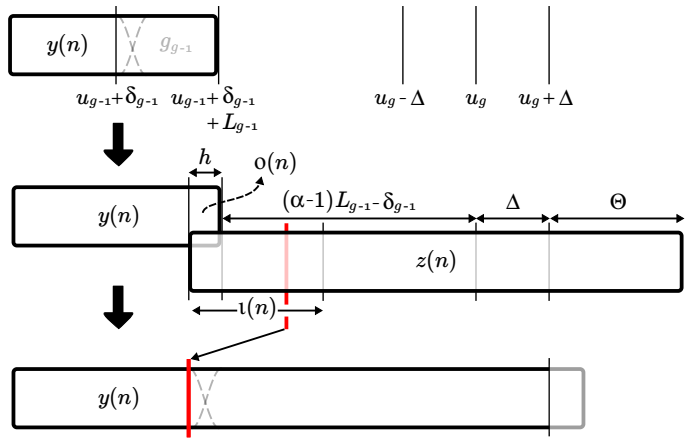

Figure 6.14. *Z samples of a signal $z(n)$ are synthesized and overlap-added at the end of $y(n)$. A cross-correlation between $o(n)$ and $\iota(n)$ is used to select the part of $z(n)$ that minimizes discontinuities in the overlap-add region. The extra samples at the end of $z(n)$ (after $u_g + \Delta$) will be discarded when inserting grain g_g as shown in Figure 6.5 of Section 6.3.1.*

8192, $\lambda = 0.25$ and $\Theta = 4096$. All but Θ, which is set arbitrarily, have been chosen experimentally through trial and error during informal tests. Note that, according to these preliminary tests, the sensitivity of the method to changes of these values is acceptable as they can vary in a relatively large span without affecting the acoustic quality of the time-stretched audio. For instance, the most critical parameter is probably λ and it can vary with few noticeable effects, in a range $[0.15, \ldots, 0.35]$ inside its complete range of possible values $[0, \ldots, 1]$. The other parameters are set to power-of-two values in order to speed up the computations but can take other values without altering much the quality of the results. However, reducing Θ too much could limit the possibilities to find a region of $z(n)$ that overlaps with g_{g-1} well enough that the junction is inaudible. Likewise, reducing N_l will induce more and more musical noise.

Another fundamental parameter is C. Contrary to the preceding parameters, it has been found to be dependent from the type of sound that is time-stretched. For quieter sports such as baseball, tennis and cricket a value $15 \leq C \leq 25$ generally gives the best results, whereas for noisy sports like

football and rugby, a higher value of $45 \leq C \leq 55$ gives better results. It must be noted that these values are valid only for $N_l = 8192$. Indeed, the coarseness of the spectral envelopes obtained through cepstral transform depends on C/N_c, the percentage of coefficients that are retained and N_c, the total number of cepstral coefficients, is a function of N_l, with $N_c = N_l/2$.

In order to reduce the computational cost of generating the Gaussian white noise used in this process, we apply the same optimization that is described in the subsection *Results and Discussions* of Section 5.2.2. We generate a few seconds of random signal and compute and store their spectral phases beforehand. These phases are then used during the cross-synthesis process.

Results

As we can see in Figure 6.15, this method creates realistic spectrograms, as opposed to the artifact seen in Figure 6.8. We study more deeply the acoustic quality of theses results through subjective listening tests in Section 6.6 and show that it is acceptable for public broadcast.

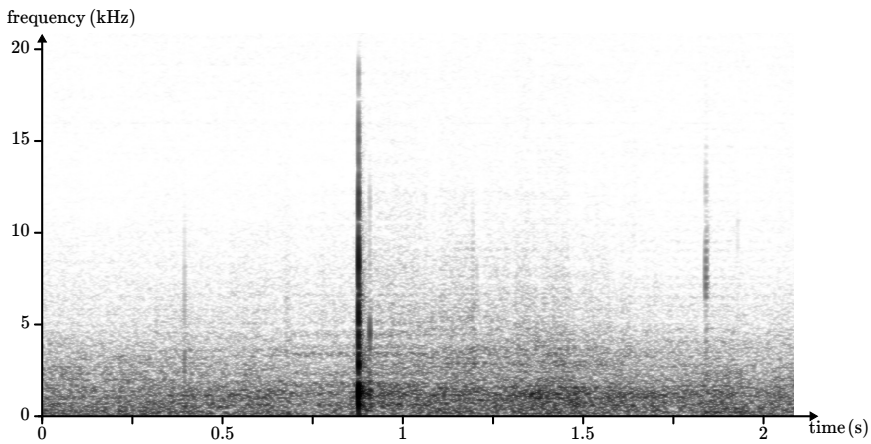

Figure 6.15. *Spectrogram of a time-stretched baseball recording (613B) already used to illustrate Figures 6.7 and 6.8. Here the self cross-synthesis method with low-pass post-filtering from Section 6.4.3 is applied. We observe that the regions around the transient do not seem to be distorted anymore by its nearby presence, as opposed to Figure 6.8.*

6.4.4 Texture

We explained in Section 5.5 how an algorithm for sound texture synthesis [71] can be turned into a time-scaling method. We used this approach to generate sound textures that fill each space left between two grains. When content is needed to fill a gap between g_{g-1} and g_g, we use other grains in their direct surrounding in the input signal $x(n)$ to generate a given length of sound texture which is then inserted in $y(n)$, after g_{g-1}, in the same way that the content generated through spectral synthesis is inserted in Sections 6.4.1 and 6.4.3.

In practice, the grains used to generate the texture are selected from $x(n)$ in a range $[t_g - M_1, \ldots, t_g + M_2]$ and only some of them are actually used. Indeed, the grains are tested with a transient detection method and the adaptive thresholding function used is set relatively low so as to discard as many transient grains as possible. Besides, only the G grains whose spectral envelopes are the closest to the spectral envelope of $x(n)$ around position t_g are used. The texture is synthesized using the method from Section 5.5 and it is inserted after g_{g+1} using a cross-correlation measure to reduce the discontinuities.

Discussions

We tested this approach with various combinations of $M_1 + M_2$, corresponding to a total time span equal to 500, 1000 and 2000 milliseconds. Usually M_2 is chosen smaller than M_1 so that the delay for a realtime implementation would remain reasonable. The resulting sounds, especially for sports such as football and rugby, have been judged as very hashed through informal tests with several listeners. Therefore, an energy normalization step has been added so that all the grains inserted have similar energy per sample but this only marginally improved the results. Another problem with this approach is that, at sudden changes between two different audio contents in the input signal, grains from both types of sounds are mixed up. This is the same problem that was noted in Section 5.5, for instance when a referee starts whistling. Besides, in regions with many transients, this methods either discards all but a few grains, and thus cannot synthesize textures of an acceptable quality, or it keeps some grains containing transients and, therefore, it adds new transients

to the signal. For instance, as already described in Section 5.5, applause parts last longer instead of being slower.[21]

As for the performance aspect, the search for the G closest grains and their combination into a texture is so computationally heavy that it barely qualifies as "realtime". Given this important drawback and the poor quality of the results, especially compared to the other approaches, the investigations on this method have not yet been pushed any further.

6.5 Variable Speed

When creating a slow motion video live, the speed factor α is not always constant. In EVS's systems, it is continuously controlled through a lever and a typical slow motion sequence usually presents, for instance, a first part played at $1 \leq \alpha \leq 1.25$ (100 to 80% of normal speed), followed by a gradual slow down to reach $\alpha = 3$ at the crucial instant of the action replay, and then the speed is progressively increased until it is back to 80-100%.

6.5.1 Speed Controlled

The various Equations involving α hereinabove can be rewritten using a variable grain-dependent speed factor α_g[22], which means that each grain can be time-scaled according to a different speed factor. This is included in our current implementation and allows us to create realistic variable speed slow motion videos. However, it is important to note that the value of α_g is updated only once per grain and thus unevenly since the length of each grain is different. Consequently, it would ignore some of the commands from the control lever or react with a delay due to the asynchronicity of the two processes.

This can lead to a perceptible offset between the two streams, audio and image, and therefore, for the time being, we inverted the controller-controlled relationship. We apply a time-scaling to the image stream that corresponds exactly to the asynchronous speed command applied to the audio, thus preserving, within acceptable margins, the synchronization between the two. Although it allows us to create variable speed slow motion videos with correct sound synchronization, this solution is used as a proof-of-concept and is obviously not

[21] It may prove interesting for some other applications, but not from our current perspective.
[22] To be accurate, in Equation 6.8, and its derivatives, it is in fact α_{g-1} that must be used.

usable within EVS's system as it is the audio time-scaling that should answer the command from the image slow motion, and not the other way around. However, solving this problem and integrating the method within EVS's slow motion solution is among the planned improvements to our method.

6.5.2 Position Controlled

Another approach to tackle this issue is to directly control the theoretical position u_g of each grain in the output, as a function of the command given by the control lever, as opposed to u_g being a function of u_{g-1}, α_{g-1} and L_{g-1}. This guarantees a permanent synchronization between audio and video, and the impact on the method is relatively limited. It invalidates Equation 6.8 and, consequently, Equations 6.21, 6.22 and 6.27, used in the context of the "concatenation" case. Therefore, in order to determine whether a concatenation is possible, one uses directly Equation 6.20, or its simplified form, Equation 6.49

$$|u_{g-1} + \delta_{g-1} + L_{g-1} - u_g| \leq \Delta \tag{6.49}$$

If this condition is respected, g_g is concatenated and δ_g is computed from Equation 6.26 in which it is the only unknown.

However, an issue arises with this method. Indeed, the condition in Equation 6.49 cannot be tested as long as the control system does not send the value for u_g. If the value is not known by the time the last valid samples of $y(n)$ (i.e. samples up to $y(u_{g-1} + L_{g-1} - h)$) have been sent to the audio output stream, the algorithm must start synthesizing gap-filling content, using one of the method in Section 6.4. Then, when u_g is finally sent by the control system, g_g must obviously be overlap-added since concatenation is not possible anymore. Waiting until u_g is known before outputting the audio samples that are after g_{g-1} is not an option as it may create a perceptible delay between the image and sound streams.

A solution to this problem is most probably to combine speed and position control, anticipating a temporary value of u_g as a function of the last known speed (a function of u_{g-2}, L_{g-2} and u_{g-1}), and to act accordingly. This has not been investigated further yet but is part of future developments.

6.6 Results

In this section we present the different results obtained based on the new method described in this chapter. A first result presented in Section 6.6.1 is a C++ implementation of *slowdio* that, when combined with the video extraction tools developed in Chapter 4, allows us to create slow motion videos with embedded audio. Although most of the research has been assessed through informal viewings of these slow motion videos by experts from the domain of sports broadcast, we also measured and compared the quality of our slowdio method through the more conventional MOS and CMOS subjective tests whose results are detailed in Section 6.6.2.

6.6.1 Implementation

We wrote a C++ library that incorporates all the signal processing tools from Chapter 1, needed to create a *slowdio*. This library can manipulate frames from a sound and compute their energy, FFTs, cepstrums, their respective inverse transforms, and so on. It can also compute an FFT-based cross-correlation between two frames and perform windowing and overlap-add. It has been used to implement the algorithm described in this Chapter, picking the following options: energy-based segmentation with search for nearby close-to-zero sample, cross-correlation-based grain shift with possible concatenation and cepstral self cross-synthesis to fill the inter-grain spaces.

This program implementing the slowdio method has been used to generate all the slow motion videos used in the subjective tests of Section 6.6.2.

Performance

We measured the average execution time of the slowdio method for the same tennis file 610C as in Table 6.1. It has 1,228,800 samples which corresponds to 25.6 seconds at 48 kHz. Table 6.2 presents the average results obtained over 5 runs of the application for different common values of α. The processor is an Intel Core 2 Duo P9700 at 2.80GHz (only one core was used) and the operating system is Linux (Ubuntu 10.04 64 bits).

Table 6.2. *Average runtime of the C++ application as a function of α for a file duration of 25.6 seconds. Values of α are typical of sports slow motion videos and correspond respectively to $^{24}/_{25}$, 90%, 80%, $^{2}/_{3}$, 50%, 33%, 25% and 20%. "% (in)" gives the relative runtime of the process as a percentage of the input duration (25.6 s). "% (out)" gives the relative runtime of the process as a percentage of the output duration ($\alpha \cdot 25.6$ s).*

α	1.042	1.11	1.25	1.5	2	3	4	5
$\bar{t}$ (s)	1.358	2.104	3.324	5.01	7.084	9.226	10.002	10.498
% (in)	5.3	8.2	13.0	19.6	27.7	36.0	39.1	41.0
% (out)	5.1	7.4	10.4	13.0	13.8	12.0	9.8	8.2

About two-thirds of the processing time is spent in the self cross-synthesis step and the generation of $Z(n)$, the remainder corresponds to the computation of cross-correlations and sample manipulations (windowing, overlap-add, etc.).

Note that the current implementation of the algorithm is not realtime, in the sense that it is not interactive and does not answer commands from users during runtime. It processes one input file to create a time-scaled version of it as its output. However, the results from Table 6.2 show that it could perform the same operation in a realtime context, as far as the computational cost is concerned. Indeed, the duration of one run is smaller than that of both the original recording and its time-scaled version. In other words, the process has more time available to run than necessary. For instance, to compute a signal extended by a factor $\alpha = 3$, it needs about 9.2 s whereas the playback duration of the resulting signal is 76.8 s.

Realtime implementation

In the previous section, we have shown that the computational cost of the algorithm is low enough to consider a realtime re-implementation of the method. However, computational cost is not the only factor to take into account when switching to the world of realtime sound processing.

For instance, one might rightfully argue that not all of the playback time span (e.g. 76.8 s in the above example) is available to compute the signal.

As a matter of fact, $\iota(n)$, the Θ-sample synthetic content needed to compute the correlation of Equation 6.42, must be generated in advance so that it is ready to be inserted when all samples of the preceding grain have been used. But it should not be computed too much in advance, to preserve the system responsiveness to modifications of the synthesis parameters. Good scheduling and thorough optimization of each step of the algorithm are thus necessary so that all the samples are generated and output to the sound card just in time.

A possible approach would be to compute it while the preceding grain is played. In that case, the heaviest part of the computation[23] would happen during the playback of the grains, thus limiting the actual available computation time to the total length of the grains, i.e. the initial duration of the file, 25.6 s. This is still larger than the necessary 9.2 s but, locally, the algorithm might run out of time when a grain has a short duration.

Besides a proper time scheduling, a realtime application as we envision it should be able to receive and process input audio samples "on-the-fly", and adapt itself to user's inputs as opposed to the current state where a complete file is available and processed at once.[24] Three parameters define the potential delays in slowdio. The first one is N_l since, in order to generate the gap-filling content $Z(n)$, $N_l/2$ samples need to be known in advance at time t_g before the computation can start. The other parameters are L_{max} and N as $L_{max} + N/2$ samples are needed at time t_g to compute the value of t_{g+1}. The largest of $N_l/2$ or $L_{max} + N/2$, will define the minimum time delay between the first sample received and the first sample output by slowdio.

In practice, however, the computation time of these two parts of the algorithm (*split* and *fill*) must also be taken into account. Therefore, the $N_l/2$ samples necessary for the generation of $Z(n)$ must be known several milliseconds before t_g so that the computation ends before the audio reaches t_g. To give an idea of the duration for that computation, it is usually below 20 milliseconds in our experiments. In other words, around sample $t_g - 0.02F_s$, the samples up to $t_g + N_l/2$ must be known. As for the grain extraction, the computation time

[23] Namely, a self cross-synthesis, creating at least Θ samples of $Z(n)$ and computing the cross-correlation in Equation 6.42 for positioning of $Z(n)$. The other steps of the computation (e.g. synthesizing the content of $Z(n)$ after $Z(\Theta)$, positioning the next grain, etc.) would happen during the time the synthetic content is played.

[24] Some methods, such as [73] for instance, need a whole recording to be able to process it and, as such, cannot be re-written in a realtime framework.

is much shorter, in the order of 0.1 to 0.5 millisecond per grain. Therefore, the computation could happen while the first $L_{min} - N/2$ samples of grain g_g are sent to the sound card, since L_{min} is usually set to 10 milliseconds. As a result, the *fill* step is generally the one to be studied to measure algorithmic delays, as opposed to the *split* part.

Note that this could only create a delay for the computation of the audio when a change of playback speed is applied directly to the live video stream, which is uncommon in sports broadcast, and again only if the live broadcast has no buffering and a zero-sample delay (i.e. image and sound are sent to the viewers as soon as recorded). In most cases anyway, the slow motion is a replay that highlights some past event and all samples are available beforehand, thus introducing no other delay than the lookup and access times for the samples in the recording (playing the first grain while the synthetic samples are computed in the background). Moreover, even in the case of a live slow motion, the algorithm would quickly accumulate enough samples from the live stream since it is "using" the samples at a slower rate than they are arriving in the system. Therefore, the problem could likely be dealt with by beginning with a relatively small value of N_l and increasing it progressively.

Variable speed

The program has also been successfully adapted to create slow motion videos with varying speed, following the approach suggested in Section 6.5.1, using a text file as the input parameter. Each line of the file contains two values: an integer representing the index g of a grain and a floating point value representing the percentage of the original speed at which g_g and the following grains must be played. For instance, a file containing

```
0    100.0
50   80.0
100  50.0
150  33.33
250  50.0
300  80.0
...
```

means that the first 50 grains are played at normal speed, $\alpha = 1$, that the speed ratio of each grain from the 51^{st} until the 100^{th} is set to $\alpha = 1.25$, and so on for the next lines of the file. In order to be of any practical use, this

requires that the segmentation into grains is known beforehand and as such it can only be used in a non-interactive framework. However, this is a first step towards a more responsive interface controlling playback speeds "on-the-fly".

6.6.2 Listening Tests

During the research, most of the assessment has been performed in an informal way. Slow motion videos with time-stretched audio have been regularly created, using the excerpts listed in Chapter 4, and sent to EVS. These videos have then been presented to selected experts in the field of sports broadcast (sound engineers, directors, etc.), notably during conferences such as the International Broadcasting Convention (IBC). This allowed us to obtain a direct feedback from actual potential customers about the qualities and defects of the various methods tested as well as the commercial viability of embedding an audio channel in a slow motion video.

However, in order to measure in a more formal manner the subjective appreciation of the different methods by viewers, we performed two series of tests. The first one measures the mean opinion scores (MOS) of the time-scaled sounds generated by either the slowdio method or a phase vocoder with automatic transient detection and processing. The second test is a comparative MOS (CMOS) between slowdio and that phase vocoder. For slowdio, we use the C++ implementation from Section 6.6.1. In the phase vocoder approach, we set an analysis frame length of 16,384 samples with a frame shift of 4096 samples and transients are detected using the multi-band spectral flux as described in Sections 2.6.2 and 5.4.3, with $N = 16$ and $\lambda = 10$. Once detected, transients are processed using the *transient removal* method from Section 2.6.4. In the following, the term "phase vocoder" is used loosely and implies a step of transient detection and processing.

Mean Opinion Scores

For each test, viewers are presented with two videos: an original sport recording, used as a reference, and its slow motion version. They are asked to judge the quality of the audio in the slow motion video on a scale between 0 (very bad) and 5 (excellent). The slow motion is randomly chosen among three possible speeds ($\alpha = 1.5$, 2 or 3) and the audio is processed using either slowdio or a phase vocoder; which process is used is chosen randomly. Fifteen viewers

participated and each took 20 tests whose results are shown in Tables 6.3 and 6.4, and Figure 6.16.

Table 6.3. *average MOS results (with 0.95 confidence intervals), as a function of α, for the phase vocoder (ph. voc.) and the slowdio method.*

MOS	overall	$\alpha = 1.5$	$\alpha = 2$	$\alpha = 3$
ph. voc.	3.3 ± 0.18	3.4 ± 0.25	3.6 ± 0.35	2.9 ± 0.3
slowdio	3.5 ± 0.16	3.9 ± 0.22	3.8 ± 0.23	2.9 ± 0.27

Results considered sport-by-sport in Table 6.4 are consistent with the overall results from Table 6.3, with the exception of baseball which obtains an over-average score of 4.4 using slowdio and tennis which get an under-average score of 2.3 when using the phase vocoder. Note also that informal tests have pointed that slowdio seems perceptually transparent or nearly transparent for $\alpha \leq 1.25$ whereas the phase vocoder can still produce reverberation and smearing. This may prove a solid advantage, notably in conversion of videos between the 24 and 25 frame per second formats.

Table 6.4. *average MOS results (with 0.95 confidence intervals), as a function of the type of sport, for the phase vocoder (ph. voc.) and the slowdio method.*

MOS	baseball	basketball	cricket	football	hockey	tennis
ph. voc.	3.9 ± 0.4	3.1 ± 0.3	3.5 ± 0.4	3.6 ± 0.3	3.4 ± 0.3	2.3 ± 0.5
slowdio	4.4 ± 0.3	3.3 ± 0.4	3.4 ± 0.3	3.3 ± 0.3	3.5 ± 0.4	3.4 ± 0.5

Comparative Mean Opinion Scores

For each test, viewers are presented with three videos: an original sport recording, used as a reference, and two slow motions A and B, with the same speed factor randomly chosen among $\alpha = 1.5$, 2 or 3. One of A or B is processed using slowdio, the other with the same phase vocoder as in the MOS test, which one is A or B is random. Viewers are asked to choose which video has

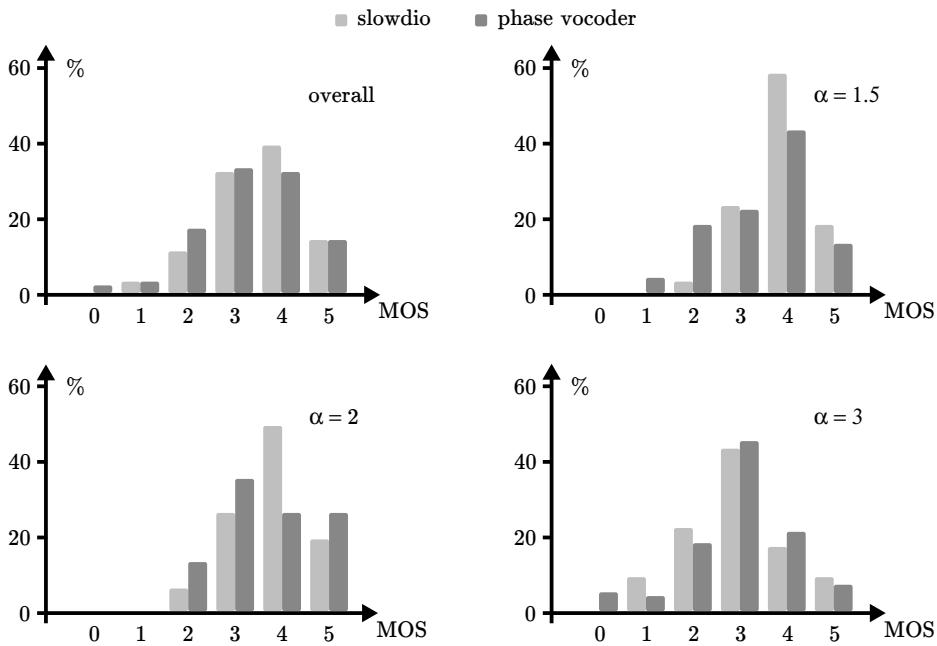

Figure 6.16. *Normalized histograms of the answers to the MOS tests globally and as a function of* α*. We can see that the results for both* slowdio *and the* phase vocoder *are similar, with a small overall preference for slowdio (more "4", less "2", no "0"). However, this preference decreases as* α *increases. For the two methods, most of the results fall in the range 3-4 and are concentrated above the threshold of* poor quality *(2).*

the best audio quality and how much better it is on a three-point scale (*slightly better*, *better* or *much better*); the option "They are the same" is also provided. Fifteen viewers participated and each took 10 tests whose results are shown in Tables 6.5 and 6.6, respectively as a function of α and as a function of the sport considered.

Similarly to the MOS test, we observe on the histograms of Figure 6.17 that slowdio is globally considered as slightly better than the phase vocoder. Even though the mean and confidence intervals from Table 6.5 indicate that the advantage can be considered as significant overall and for $\alpha = 1.5$ and 3, it is not as large a preference as we expected. Indeed, it does not seem to

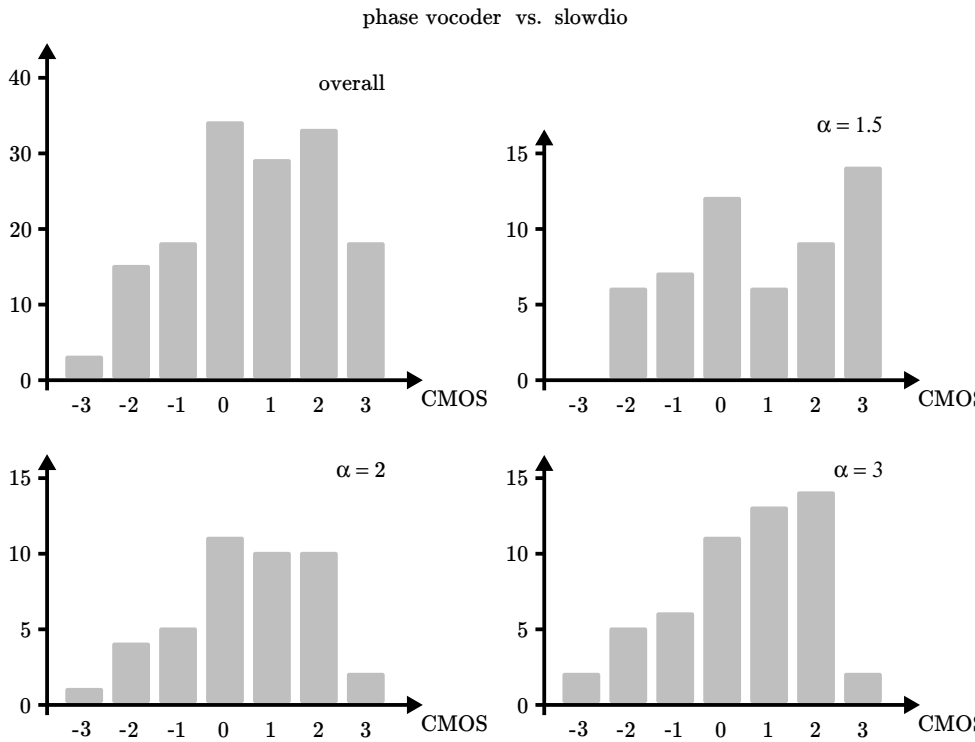

Figure 6.17. *Histograms of the answers to the CMOS tests globally and as a function of α. Positive values on the horizontal axis (1, 2 and 3) correspond to a preference for slowdio, respectively* slightly better, better *and* much better, *as given by the viewers. Likewise, negative values indicate a preference for the phase vocoder. We can observe a small but clear bias in favor of slowdio (e.g. the small amount of -3 overall).*

fit what we observe, especially in the histogram for $\alpha = 3$ which exhibits a relatively strong bias in favor of slowdio. The skewness parameter for that histogram, also given in Table 6.5 confirms the asymmetry in favor of slowdio and, therefore, a possible inadequacy of the normal model.

Results considered sport-by-sport in Table 6.6 show a slight preference for slowdio over the phase vocoder for football and tennis recordings. Slowdio is preferred for the other sports as well but not significantly.

Table 6.5. *average CMOS results (with 0.95 confidence intervals) as a function of α. The range of values is $-3 \leq \text{CMOS} \leq 3$. A positive value means that slowdio was preferred over the phase vocoder. Medians and skewnesses are also given as indicators of the bias of the histograms of Figure 6.17.*

CMOS	overall	$\alpha = 1.5$	$\alpha = 2$	$\alpha = 3$
$\mu \pm c.i._{0.95}$	0.6 ± 0.25	0.9 ± 0.44	0.4 ± 0.42	0.5 ± 0.4
median	1	1	1	1
skewness	-0.27	-0.18	-0.38	-0.53

Table 6.6. *average CMOS results (with 0.95 confidence intervals) as a function of the type of sport. The range of values is $-3 \leq \text{CMOS} \leq 3$. A positive value means that slowdio was preferred over the phase vocoder.*

CMOS	baseball	basketball	cricket	football	hockey	tennis
$\mu \pm c.i._{0.95}$	0.3 ± 0.6	0.6 ± 0.7	0.2 ± 0.5	0.8 ± 0.5	0.6 ± 0.8	1.2 ± 0.6
median	0	0	0	1	0.5	2
skewness	-0.5	0.1	$-5 \cdot 10^{-5}$	-0.2	0.08	-1.34

Informal Discussions

Informal discussions with viewers highlighted that slowdio handles transients much better, as expected, and is favored over the phase vocoder whenever a visually significant event (shoot, goalkeeper save, tennis service, etc.) is missed or mishandled by the transient processing integrated within the phase vocoder. However, it can also present discrepancies, such as a residual musical noise in crowd sounds or ripples in speech parts, when compared to the phase vocoder. Note that with $C = 0$, the method is equivalent to spectral synthesis as described in Section 6.4.1. As such, the quality of background noises is comparable to that of the phase vocoder, but causes artifacts in synthetic content inserted between grains around transients as shown in Figure 6.8.

6.7 Conclusions

In this chapter we presented a new method for time-scaling of audio recordings from sports events, in order to add an audio channel to slow motion videos. The method, named *slowdio*, combines time-domain and spectral-domain or model-based approaches. Tests have shown that the quality is acceptable for the viewers and similar or slightly superior to that obtained with state of the art approaches. Contrary to these, our approach is not based on sinusoidal and pseudo-periodical hypotheses and preserves transient events from the input signal without having to actually detect them. However, the results for background noises need to be improved, notably for large time-stretching factors. The algorithm is fast enough to consider a realtime interactive implementation for which various delays and the amount of samples needed in advance have to be carefully considered. However, slow motion videos are usually a playback of a past action for which all audio samples are already available. As a consequence, the only delays to be actually considered would be the duration of each computation.

Conclusions

In this work we described the various aspects of our research on time-scaling carried out in collaboration with EVS Broadcast Equipment SA. Time-scaling is the process that enables us to modify the duration of an audio signal without altering its acoustic content. The project has focused mainly on extending the duration of sports recordings to add an acoustic dimension to slow motion videos which are as silent today as they are omnipresent. In our scenario, the goal is that events such as ball impacts, footsteps, crowd noises, whistles, etc. remain simultaneously synchronized with the image stream and perceptually identical to the sound in the original video. Or, at the very least, they should not be perceived by the viewers as missing or distorted.

Besides two opening chapters dedicated to digital audio signal processing and to a state of the art of time-scaling for audio recordings, four new contributions related to time-scaling have been presented in as many chapters:

- PVSOLA, a new method for time-scaling of harmonic signals
- a database of sports recordings, partly annotated
- a study of the behavior of state of the art methods for time-scaling when applied and adapted to sports recordings
- *slowdio*, a new approach developed specifically for sports recordings and that implicitly preserves all the transient events

The **first contribution**, PVSOLA, has not been developed directly in relation with the problem of extending the duration of sports recordings. However, it is a new method that improves the acoustic quality of time-scaling when

applied to harmonic signals such as speech or singing. It combines a frequency-domain approach, the phase vocoder, with a time-domain one, SOLA. This significantly reduces the *phasiness*, a common artifact of the phase vocoder, and is preferred by listeners to various related time-scaling methods.

The **second contribution** has been to assemble, aurally and visually analyze, and partly annotate a database of video recordings of various sports. To the best of our knowledge, no such database existed beforehand and ours contains 229 excerpts or about 1.5 hour of various popular sports. Moreover, using software libraries put at our disposal by EVS, several tools have been developed. With these, we can process the videos, extract audio and image streams, and create new slow motion excerpts. The resulting videos with time-scaled audio have notably been demonstrated at IBC.

The **third contribution** has consisted in studying the effect of applying various existing time-scaling methods to sports recordings, which do not fit the underlying hypothesis of these algorithms. Despite this, we have tried to adjust the different parameters of each approach to obtain the best possible results. In many cases, such as time-domain and model-based approaches, the time-scaled audio signals exhibit poor quality or the processes have a high sensitivity to their parameters, hence requiring fine tuning of these. This strongly reduces the possibility to generalize their use to the huge variety of recording conditions encountered in sports broadcast.

Nevertheless, we have obtained interesting results with the phase vocoder (or, alternatively, the spectral synthesis method based on the modification of a Gaussian noise spectral amplitude through STFT), especially when combined with proper processing of transient events. The detection of these transients has proved to be challenging, especially for sports where the level of background noise makes it all but impossible to detect them all correctly. Besides, in some cases, the density of transient events is too high to be able to process them all accurately anyway. However, as we pointed out in the first paragraph, what is important is how viewers perceive the videos. As a matter of fact, subjective tests have shown that a good enough detection usually satisfy viewers, as long as the most visually significant transients are kept intact and correctly synchronized with the images in the video stream. Likewise, we observe strong rejection from the viewers in case an important transient goes missing or is exaggeratedly distorted.

With our **fourth and last contribution**, we have tried to address these issues. We introduced *slowdio*, a new approach based on realistic hypotheses about the audio in sports recordings. It produces pleasant time-scaled sounds while preserving all the transients and fine details of the original recordings. Its main advantage over the previous methods is that it does not need a transient detection step as it implicitly handles them in the time-domain with no modifications apart from their shifted time position.

The principle of this new method can be decomposed in three steps "split-shift-fill". In a first step, the input audio is divided into non-overlapping frames of variable size, called grains. The boundaries of these grains are put at minimums of energy or spectral flux of the signal to ensure that no transients are harmed in the process. In a second step, the grain positions are shifted to match the required speed factor. Then, if the shift process has created gaps between grains, these are filled with content that can be generated with various methods, such as spectral-domain, model-based or texture-based approaches. The actual position of each grain in the output is eventually adjusted around its new location to minimize discontinuities while preserving sufficient synchronization with the video stream.

Filling the inter-grain gaps generates time-scaled audio of excellent quality in stationary parts of the signal but the long-term analysis frames required to produce an acceptable acoustic quality cause artifacts around important transients that get enclosed within those analysis frames. We proposed a new approach, named self cross-synthesis, that combines short-term and long-term analysis to create spectral amplitudes with reduced transients interferences.

Finally, we compared our method to a phase vocoder with transient processing and obtained slightly better or comparable results while not requiring transient detection. Two series of tests have been run, one that independently measures the quality of each method compared to the original video recordings (MOS tests) and one that directly compares the two methods with each other (CMOS tests). *Slowdio* systematically performed slightly better with regard to the mean scores and histograms, although only a few cases showed actual statistical significance when considering confidence intervals. Further informal discussions have pointed out that our method is appreciated by the viewers for the quality of the transients and lesser reverberation whereas the time-scaling provided by the phase vocoder with transient processing is preferred for its smoother background noise.

Future Works

In the continuation of the collaboration with EVS, a public-private partnership has begun mid-2013. Its goal is to develop a version of *slowdio* that can eventually be interfaced within EVS's systems. In order to do so, different aspects of the current system have to be worked on.

First, the algorithm has to be implemented and optimized in a realtime and interactive framework so that it can be tested in real-life production environments. This implies to study the different computing bottlenecks of our approach and to find the best scheduling for each task over time to ensure a continuous and seamless time-scaled audio signal while featuring instant reactivity to speed variations. We will also work on improving the overall acoustic quality, especially reducing the ripples in the synthetic background noises, and the generalization of the *slowdio* method to new sports. Indeed, until now only the most popular sports have been tested, but time-scaled audio could be an interesting extension for others.

A major improvement that has not been considered in this thesis is time-scaling of stereophonic signals. Spatialized audio, in particular, requires a special attention as the slightest change of timing between the audio channels can destroy the spatialization effect. The shift allowed for each grain around its theoretical position and the occasional amplitude inversion in case of negative correlation will be as many challenging issues to the preservation of spatialization.

Finally, we plan to make the opposite journey that led us to *slowdio* and try to adapt its reasoning to time-scaling of speech signals. Preliminary tests have shown that, although the time-domain part of the algorithm (the grain extraction and displacement) fits perfectly with speech-like signals, the spectral-domain part is currently ill-adapted as it creates noisy segments in otherwise clean speech. However, for small ratio ($\alpha < 1.25$), *slowdio* already produces results of reasonable quality, although not completely clean, that let us think it is an interesting path to follow. For instance, using synthetic samples generated by a phase vocoder could give improved results but self cross-synthesis cannot reduce transient interferences in the phase component of the spectrum.

Appendix A

Mel-Spaced Filter Bank

A.1 Mel Scale

The mel scale is a non-linear mapping of the frequency axis that matches the human perception of these frequencies better than their linear distribution. Different mappings exist, but the most commonly used is

$$m = \text{mel}(f) = 2595 \log_{10}(1 + \frac{f}{700}) \tag{A.1}$$

$$f = \text{mel}^{-1}(m) = 700(10^{\frac{m}{2595}} - 1) \tag{A.2}$$

where f is the frequency value, in hertz, and m its corresponding number of mels. The formula is built so that a frequency of 1000 Hz corresponds exactly to 1000 mels. Note that for frequencies below 1000 Hz the mapping is often replaced by a linear one, $m = f$, since the human perception of frequencies is closer to linear than logarithmic below 1 kHz [90].

A.2 Filter Bank

A mel-spaced filter bank is a bank of B passband filters whose center frequencies are spaced apart non-linearly following the logarithmic mel-scale distribution. In other words, the centers of the B filters are spaced linearly in the mel domain. The shape we use for the filters is the triangle, as illustrated in Figure A.1, although other shapes, such as raised cosines, could be used. Each filter band starts at the center frequency of the previous filter and ends at the

center frequency of the next one. The position of the center frequency f_b of each filter is computed using Equations A.3 and A.4.

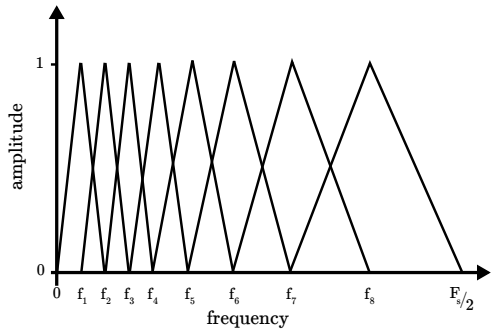

Figure A.1. *Example of triangular filter bank for $B = 8$ with frequencies spaced apart following a logarithmic progression.*

$$\Delta m = \frac{\text{mel}(\frac{F_s}{2})}{B+1} \tag{A.3}$$

$$f_b = \text{mel}^{-1}(b\Delta m) \tag{A.4}$$

with $b = [1, \ldots, B]$. Note that $f_0 = 0$ and $f_{B+1} = {}^{F_s}/_2$, but these do not correspond to the central frequencies of any filter. They are only used to compute the first and last filters of the filter bank $\Omega(b, k)$ in Equation A.5

$$\Omega(b,k) = \begin{cases} 0 & \text{if } 0 \le k \le k_{b-1} \\ \frac{k-k_{b-1}}{k_b-k_{b-1}} & \text{if } k_{b-1} \le k \le k_b \\ \frac{k-k_{b+1}}{k_b-k_{b+1}} & \text{if } k_b \le k \le k_{b+1} \\ 0 & \text{if } k_{b+1} \le k \le k_{B+1} \end{cases} \tag{A.5}$$

where k is a frequency bin of the discrete Fourier transform (DFT), $k_b = N\frac{f_b}{F_s}$, and N is the total number of points of the DFT on which we want to apply the filter bank. Note that the number of points of the bank as computed in Equation A.5 is equal to ${}^N/_2 + 1$ as we usually work only on the first half of the DFT since it is symmetrical.

Bibliography

[1] C. E. Shannon, "Communication in the presence of noise", *Proc. Institute of Radio Engineers*, vol. 37, no. 1, pp. 10–21, January 1949, reprint as Classic Paper in: Proc. IEEE, Vol. 86, No. 2, (February 1998).

[2] A. D. Götzen, N. Bernardini, and D. Arfib, "Traditional (?) implementations of a phase-vocoder: the tricks of the trade", in *Proc. of the 3rd International Conference on Digital Audio Effects (DAFx-00)*, Verona, Italy, December 7–9, 2000, pp. 37–44.

[3] F. J. Harris, "On the use of windows for harmonic analysis with the discrete fourier transform", *Proceedings of the IEEE*, vol. 66, no. 1, pp. 51–83, January 1978. [Online]. Available: http://dx.doi.org/10.1109/PROC.1978.10837

[4] S. J. Orphanidis, *Introduction to Signal Processing*. Prentice Hall, Upper Saddle River, NJ 07458, August 1995.

[5] R. Boite and M. Kunt, *Traitement de la parole*. Presses polytechniques romandes, 1987, ch. Traitement Numérique des Signaux, pp. 15–53.

[6] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*. Prentice–Hall, 1975, ch. The Discrete Fourier Transform, pp. 87–135.

[7] ——, *Digital Signal Processing*. Prentice–Hall, 1975, ch. Computation of the Discrete Fourier Transform, pp. 284–336.

[8] B. Gold and N. Morgan, *Speech and Audio Signal Processing – Processing and Perception of Speech and Music*. John Wiley & Sons, Inc., 2000, ch. Digital Filters and Discrete Fourier Transform, pp. 83–102.

[9] H. Sorensen, D. Jones, M. Heideman, and C. Burrus, "Real-valued fast fourier transform algorithms", *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 35, no. 6, pp. 849–863, June 1987.

[10] J. B. Allen and L. R. Rabiner, "A unified approach to short-time fourier analysis and synthesis", *Proceedings of the IEEE*, vol. 65, no. 11, pp. 1558–1564, November 1977.

[11] J. O. Smith, *Spectral Audio Signal Processing*. https://ccrma.stanford.edu/~jos/sasp/, accessed 2012-12-03, online book.

[12] D. Gabor, "Theory of communication. part 1: The analysis of information", *Journal of the Institution of Electrical Engineers - Part III: Radio and Communication Engineering*, vol. 93, no. 26, pp. 429–441, November 1946.

[13] D. W. Griffin and J. S. Lim, "Signal estimation from modified short-time fourier transform", *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 32, no. 2, pp. 236–243, April 1984.

[14] B. Bogert, M. Healy, and J. W. Tukey, "The quefrency alanysis of time series for echoes: Cepstrum, pseudo-autocovariance, cross-cepstrum and saphe cracking", in *Proc. Symp. on Time Series Analysis*, 1963, pp. 209–243.

[15] A. V. Oppenheim and R. W. Schafer, "From frequency to quefrency: a history of the cepstrum", *IEEE Signal Processing Magazine*, vol. 21, no. 5, pp. 95–106, September 2004.

[16] ——, *Digital Signal Processing*. Prentice–Hall, 1975, ch. Homomorphic Signal Processing, pp. 480–531.

[17] P. Mermelstein, "Distance measures for speech recognition–psychological and instrumental", in *Proceedings of the Joint Workshop on Pattern Recognition and Artificial Intelligence*, Hyannis, Massachusetts, USA, June 1–3, 1976, pp. 374–388.

[18] P. Strobach, *Linear Prediction Theory – A Mathematical Basis for Adaptive Systems*. Springer-Verlag, 1990, ch. The Linear Prediction Model, pp. 13–36.

[19] T. Dutoit, N. Moreau, and P. Kroon, *Applied Signal Processing – A*

Matlab-Based Proof of Concept. Springer Science+Business Media, 2009, ch. How is speech processed in a cell phone conversation ?, pp. 1–31.

[20] J. Laroche, *Applications of Digital Signal Processing to Audio and Acoustics*. Kluwer Academic, 1998, ch. Time and Pitch Scale Modification of Audio Signals, pp. 279–309.

[21] J. Bonada, "Audio time-scale modification in the context of professional audio post-production", Universitat Pompeu Fabra, Barcelona, research work for PhD program, Fall 2002.

[22] R. McAulay and T. Quatieri, "Speech analysis/synthesis based on a sinusoidal representation", *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 34, no. 4, pp. 744–754, August 1986.

[23] T. Dutoit and J. Laroche, *Applied Signal Processing – A Matlab-Based Proof of Concept*. Springer Science+Business Media, 2009, ch. How does audio effects processor perform pitch shifting ?, pp. 149–185.

[24] S. Roucos and A. M. Wilgus, "High quality time-scale modification for speech", in *Proc. of International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Tampa, Florida, USA, April 26–29, 1985, pp. 493–496.

[25] W. Verhelst, "Overlap-add methods for time-scaling of speech", *Speech Communication*, vol. 30, no. 4, pp. 207–221, April 2000.

[26] D. Hejna and B. Musicus, "The SOLAFS time-scale modification algorithm", BBN Technical Report, Tech. Rep., July 1991.

[27] E. Moulines, F. Charpentier, and C. Hamon, "A diphone synthesis system based on time-domain prosodic modifications of speech", in *Proc. of International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Glasgow, Scotland, May 23–26, 1989, pp. 238–241.

[28] G. Fairbanks, W. Everitt, and R. Jaeger, "Method for time or frequency compression-expansion of speech", *Transactions of the IRE Professional Group on Audio*, vol. 2, no. 1, pp. 7–12, January 1954.

[29] D. Gabor, "Theory of communication. part 3: Frequency compression and expansion", *Journal of the Institution of Electrical Engineers - Part III: Radio and Communication Engineering*, vol. 93, no. 26, pp. 445–457, November 1946.

[30] P. Dutilleux, G. D. Poli, A. von dem Knesebeck, and U. Zölzer, *DAFX: Digital Audio Effects, Second Edition*. John Wiley & Sons, Ltd., 2011, ch. Time-segment processing, pp. 185–217.

[31] F. Charpentier and E. Moulines, "Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones", in *Proc. of the First European Conference on Speech Communication and Technology (Eurospeech)*, Paris, France, September 27–29, 1989, pp. 2013–2019.

[32] D. J. Hejna, B. R. Musicus, and A. S. Crowe, "Method for time-scale modification of signals", Patent US 5 175 769, July 23, 1991. [Online]. Available: http://www.google.com/patents?id=_igoAAAAEBAJ

[33] W. Verhelst and M. Roelands, "An overlap-add technique based on waveform similarity (WSOLA) for high quality time-scale modification of speech", in *Proc. of International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 2, April 1993, pp. 554–557.

[34] G. Pallone, "Dilatation et transposition sous contraintes perceptives des signaux audio : Application au transfert cinéma-vidéo", Ph.D. dissertation, École Doctorale de Mécanique, Physique et Modelisation, Université d'Aix-Marseille II, pp. 72–73, 2003.

[35] E. Moulines and F. Charpentier, "Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones", *Speech Communication*, vol. 9, no. 5–6, pp. 453–467, 1990. [Online]. Available: http://www.sciencedirect.com/science/article/pii/016763939090021Z

[36] E. Moulines and J. Laroche, "Non-parametric techniques for pitch-scale and time-scale modification of speech", *Speech Communication*, vol. 16, no. 2, pp. 175–205, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/016763939400054E

[37] J. L. Flanagan and R. M. Golden, "Phase vocoder", *Bell System Technical Journal*, vol. 45, no. 9, pp. 1493–1509, November 1966.

[38] H. Dudley, "The vocoder", *Bell Labs Record*, vol. 17, no. 2, pp. 122–126, 1939.

[39] D. Tompkins, *How to Wreck a Nice Beach: The Vocoder from World War II to Hip-Hop, The Machine Speaks*. Melville House, 2010, p. 40.

[40] R. Schafer and L. Rabiner, "Design and simulation of a speech analysis-

synthesis system based on short-time fourier analysis", *IEEE Transactions on Audio and Electroacoustics*, vol. 21, no. 3, pp. 165–174, June 1973.

[41] M. R. Portnoff, "Implementation of the digital phase vocoder using the fast fourier transform", *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 24, no. 3, pp. 243–248, June 1976.

[42] M. Dolson, "The phase vocoder: A tutorial", *Computer Music Journal*, vol. 10, no. 4, pp. 14–27, Winter 1986.

[43] J. Laroche and M. Dolson, "Improved phase vocoder time-scale modification of audio", *IEEE Transactions on Speech and Audio Processing*, vol. 7, no. 3, pp. 323–332, May 1999.

[44] J. Bonada, "Automatic technique in frequency domain for near-lossless time-scale modification of audio", in *Proc. of the International Computer Music Conference (ICMC)*, Berlin, Germany, 27 August – 1 September 2000, pp. 396–399.

[45] D. P. W. Ellis, "A phase vocoder in Matlab", 2002, web resource, last consulted in March 2011. [Online]. Available: http://www.ee.columbia.edu/~dpwe/resources/matlab/pvoc/.

[46] M. Puckette, "Phase-locked vocoder", in *Proc. of IEEE ASSP Workshop on Applications of Signal Processing to Audio and Acoustics*, Mohonk, NY, USA, October 15–18, 1995, pp. 222–225.

[47] J. Laroche and M. Dolson, "Phase-vocoder: about this phasiness business", in *Proc. of 1997 IEEE ASSP Workshop on Applications of Signal Processing to Audio and Acoustics*, New Paltz, NY, USA, October 19–22, 1997, pp. 55–58.

[48] T. Karrer, E. Lee, and J. Borchers, "PhaVoRIT: A phase vocoder for real-time interactive time-stretching", in *Proc. of the International Computer Music Conference (ICMC)*, New Orleans, USA, November 6–11, 2006, pp. 708–715.

[49] D. Doran, E. Coyle, and R. Lawlor, "An efficient phasiness reduction technique for moderate audio time-scale modification", in *Proc. of the 7th International Conference on Digital Audio Effects (DAFx-04)*, London, UK, October 5–8, 2004, pp. 83–88.

[50] A. Moinet and T. Dutoit, "PVSOLA: A phase vocoder with synchronized

overlap-add", in *Proc. of the 14th International Conference on Digital Audio Effects (DAFx-11)*, Paris, France, September 19–23, 2011, pp. 269–275, [DAFx Best Student Paper Award - Bronze].

[51] S. Kraft, M. Holters, A. von dem Knesebeck, and U. Zölzer, "Improved PVSOLA time-stretching and pitch-shifting for polpyhonic audio", in *Proc. of the 15th International Conference on Digital Audio Effects (DAFx-12)*, York, UK, September 17–21, 2012.

[52] G. Fant, *Acoustic Theory of Speech Production.* The Hague: Mouton, 1960.

[53] X. Serra, "A system for sound analysis/transformation/synthesis based on a deterministic plus stochastic decomposition", Ph.D. dissertation, Stanford University, Department of Music, 1989.

[54] T. S. Verma and T. H. Y. Meng, "Time scale modification using a sines+transients+noise signal model", in *Proc. of the 1st International Conference on Digital Audio Effects (DAFx-98)*, Barcelona, Spain, November 19–21, 1998, pp. 49–52.

[55] Y. Stylianou, J. Laroche, and E. Moulines, "High-quality speech modification based on a harmonic noise model", in *Proc. of the Fourth European Conference on Speech Communication and Technology (Eurospeech'95)*, Madrid, Spain, September 18–21, 1995, pp. 451–454.

[56] Y. Stylianou, "Harmonic plus noise models for speech combined with statistical methods, for speech and speaker modifications", Ph.D. dissertation, École Nationale Supérieure des Télécommunications, 1996.

[57] A. Röbel, "A shape-invariant phase vocoder for speech transformation", in *Proc. of the 13th International Conference on Digital Audio Effects (DAFx-10)*, Graz, Austria, September 6–10, 2010.

[58] T. Karrer, "PhaVoRIT: A phase vocoder for real-time interactive time-stretching", Master's thesis, RWTH Aachen University, Aachen, Germany, November 2005.

[59] J. P. Bello, L. Daudet, S. Abdallah, C. Duxbury, M. Davies, and M. B. Sandler, "A tutorial on onset detection in music signals", *IEEE Transactions on Speech and Audio Processing*, vol. 13, no. 5, pp. 1035–1047, September 2005.

[60] S. N. Levine and J. O. S. III, "A sines+transients+noise audio representation for data compression and time/pitch scale modifications", in *Proc. of the 105th Audio Engineering Society Convention (AES)*, San Francisco, California, US, September 26–29, 1998.

[61] F. Nagel and A. Walther, "A novel transient handling scheme for time stretching algorithms", in *Proc. of the 127th Audio Engineering Society Convention (AES)*, New-York, USA, October 9–12, 2009.

[62] I. Damnjanovic, D. Barry, D. Dorran, and J. D. Reiss, "A real-time framework for video time and pitch scale modification", *IEEE Transactions on Multimedia*, vol. 12, no. 4, pp. 247–256, June 2010.

[63] P. Masri and A. Bateman, "Improved modeling of attack transients in music analysis-resynthesis", in *Proc. of the International Computer Music Conference (ICMC)*, Hong Kong, August 19–24, 1996, pp. 100–103.

[64] A. Röbel, "A new approach to transient processing in the phase vocoder", in *Proc. of the 6th International Conference on Digital Audio Effects (DAFx-03)*, London, UK, September 8–11, 2003.

[65] P. Masri, "Computer modeling of sound for transformation and synthesis of musical signal", Ph.D. dissertation, University of Bristol, 1996.

[66] N. Saint-Arnaud and K. Popat, *Computational auditory scene analysis*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1998, ch. Analysis and Synthesis of Sound Textures, pp. 293–308.

[67] L. Lu, L. Wenyin, and H.-J. Zhang, "Audio textures: Theory and applications", *IEEE Transactions on Speech and Audio Processing*, vol. 12, no. 2, pp. 156–167, March 2004.

[68] J.-J. Filatriau, "Analysis, synthesis and gestural control of expressive sonic textures in musical contexts", Ph.D. dissertation, Université catholique de Louvain, École polytechnique de Louvain, June 2011.

[69] G. Strobl, G. Eckel, and D. Rocchesso, "Sound texture modeling: A survey", in *Proc. of the Sound and Music Computing Conference (SMC)*, Marseille, France, May 18–20, 2006.

[70] D. Gabor, "Acoustical quanta and the theory of hearing", *Nature*, vol. 159, no. 4044, pp. 591–594, May 1947.

[71] J. Parker and B. Behm, "Creating audio textures by example: tiling and stitching", in *Proc. of International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 4, no. iv, May 17–21, 2004, pp. 317–320.

[72] S. Dubnov, Z. Bar-Joseph, R. El-Yaniv, D. Lischinski, and M. Werman, "Synthesizing sound textures through wavelet tree learning", *IEEE Computer Graphics and Applications*, vol. 22, no. 4, pp. 38–48, July/August 2002.

[73] C. Picard, N. Tsingos, and F. Faure, "Retargetting example sounds to interactive physics-driven animations", in *Proc. of the 35th International Conference on Audio for Games (AES)*, London, UK, February 11-13 2009. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=15179

[74] D. Schwarz, "Corpus-based concatenative synthesis", *IEEE Signal Processing Magazine*, vol. 24, no. 2, pp. 92–104, March 2007.

[75] C. Picard, "Expressive sound synthesis for animation", Ph.D. dissertation, Université de Nice - Sophia Antipolis / Ecole Doctorale STIC, Décember 2009.

[76] J. Kominek and A. W. Black, "CMU arctic databases for speech synthesis", Language Technologies Institute, School of Computer Science, Carnegie Mellon University, Tech. Rep., 2003.

[77] D. P. W. Ellis, "SOLAFS in Matlab", 2006, web resource, last consulted in March 2011. [Online]. Available: `http://www.ee.columbia.edu/~dpwe/resources/matlab/solafs-matlab.html`.

[78] V. Grancharov and W. Kleijn, *Handbook of Speech Processing*. Springer, 2007, ch. Speech Quality Assessment, pp. 83–99.

[79] T. Quatieri and R. McAulay, "Shape invariant time-scale and pitch modification of speech", *IEEE Transactions on Signal Processing*, vol. 40, no. 3, pp. 497–510, 1992.

[80] W.-H. Liao, A. Röbel, and A. W. Su, "On stretching gaussian noises with the phase vocoder", in *Proc. of the 15th International Conference on Digital Audio Effects (DAFx-12)*, York, UK, September 17–21, 2012.

[81] S. Boll, "Suppression of acoustic noise in speech using spectral subtraction", *IEEE Transactions on Acoustics, Speech and Signal Processing*,

vol. 27, no. 2, pp. 113–120, April 1979.

[82] Y. Ephraim and D. Malah, "Speech enhancement using a minimum-mean square error short-time spectral amplitude estimator", *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 32, no. 6, pp. 1109–1121, December 1984.

[83] K. Tokuda, T. Kobayashi, T. Masuko, and S. Imai, "Mel-generalized cepstral analysis — a unified approach to speech spectral estimation", in *Proc. of International Conference on Spoken Language Processing (ICSLP94)*, vol. 3, September 1994, pp. 1043–1046.

[84] E. Ambikairajah, A. G. Davis, and W. T. K. Wong, "Auditory masking and mpeg-1 audio compression", *Electronics & Communication Engineering Journal*, vol. 9, no. 4, pp. 165–175, August 1997.

[85] A. Moinet, T. Dutoit, and P. Latour, "Audio time-scaling for slow motion sports videos", in *Proc. of the 16th International Conference on Digital Audio Effects (DAFx-13)*, Maynooth, Ireland, September 2–6, 2013.

[86] ——, "Time-stretching of an audio signal", Patent EP 2 509 073, October 10, 2012. [Online]. Available: http://www.google.com/patents/EP2509073A1

[87] ——, "Time-stretching of an audio signal", Patent Application WO 2012/136 380 (PCT), October 11, 2012. [Online]. Available: http://www.google.com/patents/WO2012136380A1

[88] R. Burr and D. Lytle, "Comments on "a general method of minimum cross-entropy spectral estimation"", *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 34, no. 5, pp. 1324–1326, October 1986.

[89] C. Breithaupt, T. Gerkmann, and R. Martin, "Cepstral smoothing of spectral filter gains for speech enhancement without musical noise", *IEEE Signal Processing Letters*, vol. 14, no. 12, pp. 1036–1039, December 2007.

[90] L. Rabiner and B.-H. Juang, *Fundamental of Speech Recognition*. Prentice-Hall, Englewood Cliffs, NJ, 1993, ch. Pattern-Comparison Techniques, pp. 183–190.

List of Figures

List of Tables

This thesis was written using a modified version of the *hepthesis* style, on a Linux computer, with Kile as an editor, Inkscape to draw most of the images and git for versioning and backups. About 84 kg of fruits were necessary as well as a helluva mountain of daily spams™. And it was really fun ...

... except for the LaTeX part anyway ... grumble.

"Whatever doesn't kill me ...
... had better start running"