Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science

Master's Thesis

Time-Scale Modification Algorithms for Music Audio Signals

submitted by
Jonathan Driedger

submitted
November 3, 2011

Supervisor / Advisor
Priv.-Doz. Dr. Meinard Müller

Reviewers
Priv.-Doz. Dr. Meinard Müller
Prof. Dr. Michael Clausen

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement under Oath

I confirm under oath that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, _____ _____

(Datum / Date) (Unterschrift / Signature)

Acknowledgments

First of all I would like to express my deepest gratitude to my supervisor Meinard Müller who gave me the possibility to work on this compelling topic. I never imagined that writing my Master's Thesis could be that fun, and the fact that it was is mainly due to him. He always found the time to discuss my ongoing work with me and give me good advice.

The next person I would like to thank is Peter Grosche. His dedication to help me, and also all other students in the Multimedia Information Retrieval and Music Processing group, is incredible. I can not remember a single time he sent me away when I had some sort of question. He was a constant source of inspiration and motivation for me.

Furthermore, I would like to thank my fellow student and dear friend Thomas Prätzlich, who was writing his Master's Thesis at the same time as I did, for the hours and hours of discussions and mutual assistance. It was nice to always have somebody to talk about thesis-related problems.

I also would like to thank Verena Konz, Nanzhu Jiang and Marvin Künnemann for proof-reading my thesis and giving me a lot of feedback as well as Zuo Zhe for helping me with my experiments.

Last but not least I would like to thank my family and especially my partner Julia Hahnemann for their support throughout the time of writing this thesis.

Abstract

Sound is a phenomenon that lives in manifold dimensions. Besides pitch, volume, timbre and many others, the time is an essential component. In the field of *audio signal processing* not only the analysis of these aspects, and in particular the time aspect, but also their modification is an important issue. When sound becomes music, time becomes tempo and rhythm. Having the possibility to change the tempo and the rhythm of audio recordings is extremely helpful for example when a DJ wants to adapt the tempos of two songs. But when using analogue audio media like records or tapes, changing the playback speed of the music does not only affect the tempo, but also the pitch of the sound. This looks different in the world of digital audio recordings where algorithms were invented to decouple the time-scale of audio signals from their pitch. These are typically referred to as *time-scale modification* (TSM) algorithms. But although there exists a large variety of different algorithmic approaches to the field of TSM of audio recordings, there is no algorithm that produces artifact-free results robustly.

The goal of this thesis is therefore to analyze what makes TSM difficult and where artifacts originate from. We investigate two prominent TSM algorithms, one working in the time-domain (WSOLA) and one working in the frequency-domain (Phase Vocoder). Both algorithms produce their very own artifacts which we analyze and classify. Using the example of WSOLA, we show how to overcome a certain class of artifacts that are frequently produced by time-domain TSM algorithms at transients in audio signals. To evaluate the quality of different TSM algorithms we present the results of a listening test that we performed. Finally we integrate our knowledge about TSM algorithms into a soundtrack generation system that allows for creating euphonious transitions between audio recordings.

Contents

Chapter 1

Introduction

1.1 Time-Scale Modification of Audio Signals

Sound is a phenomenon that lives in manifold dimensions. Besides pitch, volume, timbre and many others, the time is an essential component of audio signals. Being able not only to analyze but also to modify aspects of sound is an important issue in the field of *audio signal processing*. Therefore also the modification of the time aspect, which is generally referred to as *time-scale modification* (TSM) of audio signals, has received a lot of interest. The task is visualized in Figure 1.1. When sound becomes music, time becomes tempo and rhythm. Having the possibility to change the tempo and the rhythm of audio recordings is extremely helpful for example in the field of DJing when a DJ wants to adapt the tempos of two songs. But when using analogue audio media like for example records or tapes, changing the playback speed of the music does not only affect the tempo, but also the pitch of the sound.

This looks different in the world of digital audio recordings where algorithms were invented to decouple the time-scale of audio signals from their pitch. These are typically referred to as TSM algorithms. But although there exists a large variety of different algorithmic approaches to the field of TSM of audio recordings, there is no algorithm that produces

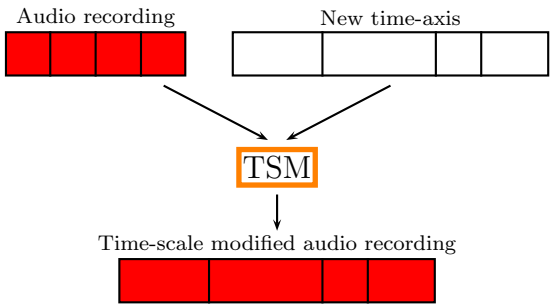

Figure 1.1. Basic principle of TSM. TSM algorithms allow for modifying the time-scale of an audio recording according to a given time-axis.

artifact-free results robustly. In this thesis we therefore analyze and explain the TSM of audio signals in general as well as prominent existing algorithms to better understand what makes TSM difficult and where artifacts originate from. In the next sections we furthermore propose a novel application that heavily relies on TSM to motivate this detailed analysis.

1.2 Motivating Application

When the first movies were created at the end of the 19^{th} century, they had no soundtrack. Nevertheless, they were almost always accompanied by musicians to emphasize the images. Nowadays it is almost impossible to imagine media like movies, computer games, TV shows, commercials, slide shows or even theater performances being presented without any audible component. But in the same way in which appropriate music can emphasize moods and emotions, inappropriate music can decrease the quality of motion pictures tremendously. Therefore the proper choice of underlaying music is extremely important in any visual media. But not only the music itself is crucial. For example at a change in the scenery of a movie the music should adapt to the images, and optimally the change in the music should be perceived as natural by the audience.

There exist various general approaches for generating soundtracks. The most common one is the *compositorial approach*. In a large movie production the music is most commonly composed, performed and recorded for this specific movie. The soundtrack is tailored to the story and any change in the scenery can be handled explicitly by the composer. Although this approach for sure yields the best results, it is extremely expensive since it involves a massive amount of creative work by the composer. Furthermore the recording process is labor-intensive and time consuming. Finally the approach is not very flexible. Once the soundtrack is recorded, greater alterations to the movie are virtually impossible. For the same reason it is difficult to apply this approach to other media like for example computer games where no strict time line of events exists. The composer has to design the music such that it is possible to repeat it over and over again in passages where the scenery does not change and at the same time offer musical transitions that bridge to the music for the next passage in the storyline of the game.

An other option is the *parametric approach*. By parameterizing music in terms of harmony, tempo, dynamics, instrumentation and further aspects it is theoretically possible to utilize a synthesizer to produce music that can adapt fast to the shown images by tuning the given parameters. In a scene of high tension the parameters could for example be set to produce loud, fast and percussive music while in the next moment change to soft and quiet music in a scene of relief. Although this approach is inexpensive and flexible it has shown that music which is completely computer-generated tends to sound aesthetically not very appealing to human listeners.

We therefore propose a novel *data-driven* approach to overcome the shortcomings of the two approaches mentioned above. Our core idea builds on the assumption that nowadays large music collections that cover a wide range of moods are widely available. From such collections suitable audio clips that correspond well to the visual scenes can be chosen and played back while accounting for user specifications. Figure 1.2 visualizes our approach.

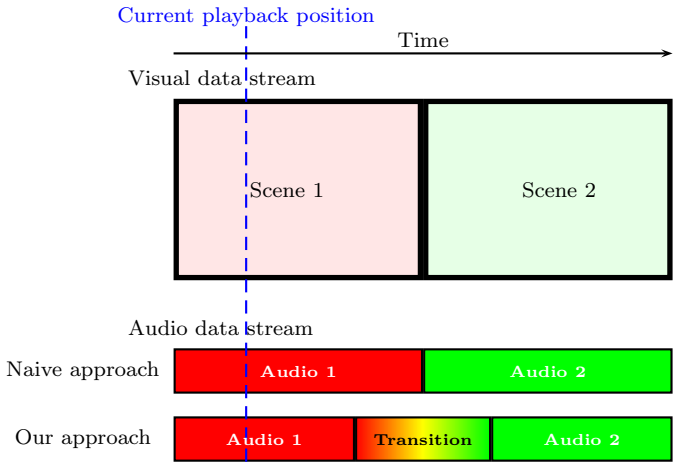

Figure 1.2. Having a collection audio recordings, the goal is to underlay a visual data stream with fitting music from this collection such that a transition between two recordings is appealing. In the naive approach we simply cut the current audio recording at the end of Scene 1 and append some recording that fits Scene 2. But this leads to abrupt breaks in the music. In our approach we are looking for an optimal recording in the set of all recording that can be used as underlaying music for Scene 2 such that we can compute an euphonious transition from the current recording to the new recording.

The problem is that a simple concatenation of audio recordings does not suffice to create an aesthetically appealing audio data stream since a change from one audio recording to the next happens abruptly and may therefore be perceived as unpleasant. The goal is therefore to create smooth transitions between two audio recordings that are as pleasant as possible to the ear of the listener.

Blending is a common technique to make the transition between two audio recordings less abrupt. By overlaying the two audio recordings by a short time and decreasing the volume of the ending recording until it is finally not perceivable any more while at the same time increase the volume of the starting recording from silence to the normal volume, a smooth transitions between the two recordings is created. Nevertheless, even when blending from one recording to another one, the transition is often still not acceptable. This is usually due to harmonic and rhythmic differences during the transition phase between the two recordings.

1.3 The Connector

To overcome those issues and to produce harmonically and rhythmically appealing transitions we developed a tool, called the Connector. Its main purpose is to connect audio recordings in a harmonic and rhythmic sensitive way such that the transition between two audio recordings is not, or only slightly noticeable. Its coarse structure is visualized in Figure 1.3. Given to the Connector are the currently playing audio recording along with a region in this audio recording were we want to switch to another recording. This region is called the *transition region*. In the first stage, the Connector looks for a region in some recording in a database that has a similar harmonic progression to the transition region.

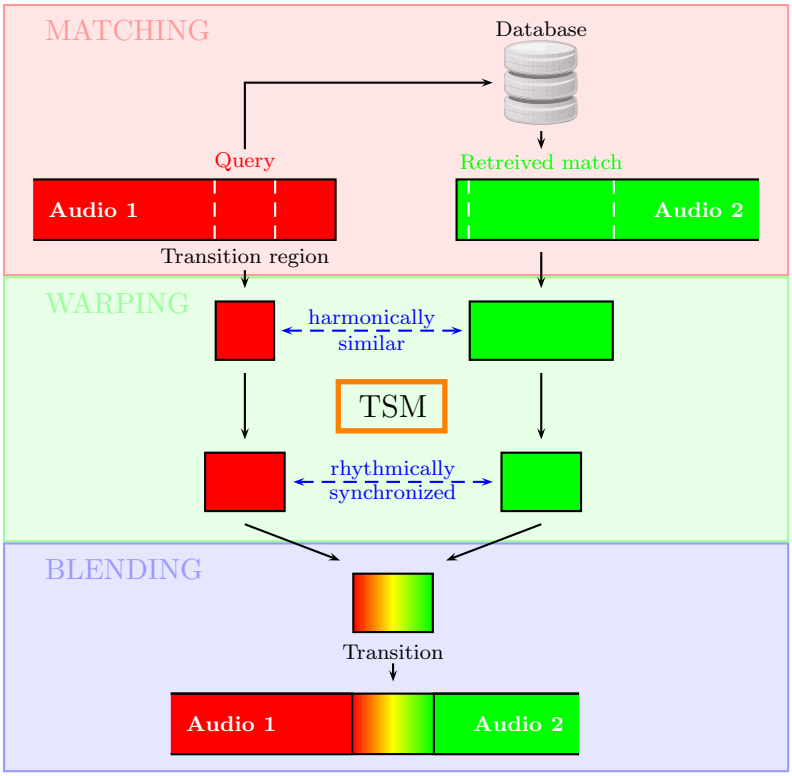

Figure 1.3. The three stages of the CONNECTOR.

The goal of this step is to assure that during the transition phase the two recordings that will be audible at the same time are harmonically related. Having found a match, the CONNECTOR rhythmically synchronizes the transition region with the matched region by aligning the beat positions of both regions temporally. This is done to avoid a chaotic sound that would result from overlaying the two regions without any adaptions. To align the beat positions, the two audio recordings are warped along the time-axis in a non-linear fashion using TSM algorithms. Since the transition between the two recordings should be aesthetically appealing it is crucial that the applied TSM algorithm introduces as little audible artifacts into the audio recordings as possible. At the end of the warping stage the two regions can be regarded both harmonically similar and rhythmically synchronized. In the last stage the warped audio recordings are finally blended to finish the transition.

1.4 Contribution

The main contribution of this thesis lies in the detailed analysis of different TSM algorithms and TSM of audio signals in general. Even though TSM techniques have received a lot of interest in recent years, non-linear TSM, which plays an essential role in the pipeline of the CONNECTOR, seems to be a topic that is often avoided in the literature. Furthermore publications in this field tend to have very diverse notations and definitions. This thesis builds a unified theoretical foundation which allows for understanding TSM algorithms in detail and also carefully addresses the topic of non-linear TSM.

Furthermore we developed and analyzed an improved version of a standard TSM algorithm. The quality of this algorithm as well as several other TSM algorithms was evaluated by performing a listening test. The results show that our modified algorithm yields time-scale modified audio recordings of significantly better quality in many cases than the standard version of the algorithm.

Finally, as a practical part of this thesis, a first prototype of the Connector as described in Section 1.3 was implemented in MATLAB.

1.5 Thesis Organization

This thesis is structured as follows. In Chapter 2 we introduce the foundations that are necessary to work with audio signals on a theoretical level and that are used frequently throughout this thesis. Chapter 3 gives a detailed introduction to the field of time-scale modification of audio signals on an abstract level, while Chapter 4 and Chapter 5 are devoted to popular TSM algorithms. In Chapter 6 we discuss our new TSM algorithm and explain it in detail. The listening test which we performed to evaluate the quality of our new algorithm as well as several other TSM algorithms is presented in Chapter 7. We give a description of the Connector in its current state and all of its stages in Chapter 8 and close this thesis with a brief outline of future work in Chapter 9.

Chapter 2

Basic Definitions, Notations and Tools

The CONNECTOR is supposed to be a tool that is capable of producing euphonious transitions from a currently played audio recording to another. To this end, it has to analyze, interpret and modify the audio recording that is currently played as well as the audio recordings in the database. These are tasks that are not easily manageable by a machine upfront. We first need mathematical formulations of what music, or in general sound, is and how a machine can work with it. As a start, we mathematically define audio signals in Section 2.1. Afterwards we focus on the harmonic content of audio signals in Sections 2.2, 2.3 and 2.4.

Note that throughout this chapter, as well as in this whole thesis we closely follow the notations and definitions of [33].

2.1 Audio Signals

We begin with the most basic object the CONNECTOR has to deal with, the *audio signal*.

Definition 2.1 *An* audio signal *is a function $f : \mathbb{R} \to \mathbb{R}$, where the domain $\mathbb{R}$ represents the time-axis and the range $\mathbb{R}$ the amplitude of the sound wave. Since all real-world audio signals are time-limited with a duration D, we define $T = [0, D) \subset \mathbb{R}$ as the domain of the time-limited signal and assume $f(t) = 0$ for $t \in \mathbb{R} \backslash T$. With the domain being $\mathbb{R}$, such an audio signal is also referred to as continuous-time (CT) signal.*

To be able to actually process audio signals with a machine, the signal needs to be transformed into a digital representation. To this end we discretize a given CT signal and therefore turn it into a *discrete audio signal*.

Definition 2.2 *A* discrete audio signal*, also called* discrete-time (DT) signal*, is a function $x : \mathbb{Z} \to \mathbb{R}$ which is defined on a discrete subset of the temporal domain of a CT signal. Since the discrete signal is time-limited as well, we analogously define $T' = [1 : N] \subset \mathbb{N}$.*

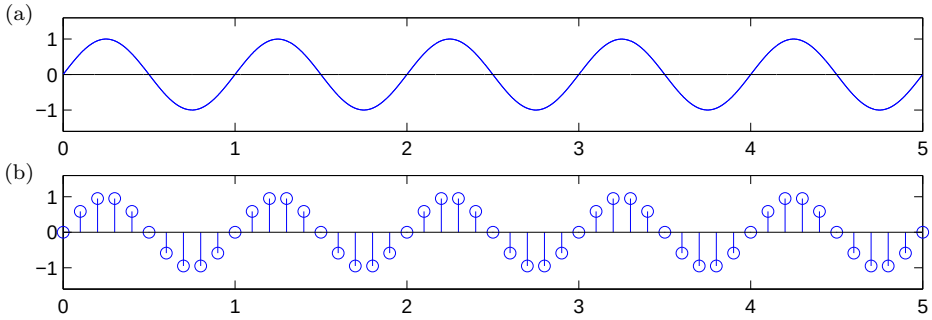

Figure 2.1. (a) A CT signal. Time is given in seconds. **(b)** The DT version of the CT signal from (a) computed by equidistant sampling with a sampling rate of 10 Hz. Time is also given in seconds.

For x being a discrete audio signal we define $length(x) = N$. Furthermore we define $x[a : b]$ for $a, b \in T'$ and $a \leq b$ to be the discrete audio signal that is the fragment of x defined on $\{z | z \geq a$ and $z \leq b\}$.

Note that in fact the range $\mathbb{R}$ of a discrete audio signal is also discretized when converting a CT signal into a DT signal. Nevertheless we will ignore this detail in the following since it does not affect our work.

A standard way to convert a CT signal f into a DT signal x is to sample the CT signal at equidistant points, which is known as *equidistant sampling*. To this end a fixed *sampling period* p_s is defined and we compute

$$x(n) = f(p_s \cdot (n-1)) \quad , \tag{2.1}$$

for $n \in \mathbb{Z}$. The inverse of the sampling period $\frac{1}{p_s}$ is commonly referred to as the *sampling rate* of the DT signal and denoted by f_s.

2.2 Pitch Features

Waveforms are the most common representation of audio signals. They simply encode the air pressure variations that the human ear perceives as sound. Nevertheless, this representation hides a lot of information. While temporal events like volume variations are well perceivable in the waveform, it is for example almost impossible to get any information about the frequency content, and therefore harmonic information about the signal from the waveform directly. To analyze an audio signal in terms of a certain aspect of sound, and therefore make this aspect accessible and interpretable for a machine, one first has to find a way to capture and quantify the desired aspect in the signal in a so called *feature*. In the context of the CONNECTOR we are especially interested in the harmonic content of audio signals. A common feature to capture this aspect of sound is the *pitch feature*. It assigns to each of the 88 MIDI pitches $p = 21$ to $p = 108$ that correspond to the keys of a standard piano a value that quantifies how present the corresponding tone is in the audio signal at a certain point in time.

To obtain a pitch feature from an audio signal x we first apply a suitable bandpass filter for each pitch p to x. This filter passes all frequencies around the center frequency of the corresponding pitch p while rejecting all other frequencies. By combining all 88 filters we get an array of filters which is called a *pitch filter bank*. The magnitude response of this filter bank yields 88 subband signals x_p for $p \in [21 : 108]$.

Since we would like to analyze the harmonic content of an audio signal at a high temporal resolution, the audio signal x is typically divided into a sequence of short overlapping segments,

$$S = S_1, S_2, ..., S_N \quad , \tag{2.2}$$

where N is the length of the sequence and it holds that all segments S_n for $n \in [1 : N]$ are of the same length. The final pitch feature is then computed by measuring the *short-time mean square power* (STMSP) in each of the subband signals x_p within each segment S_n.

$$STMSP(n, p) = \sum_{k \in S_n} |x_p(k)|^2 \quad , \tag{2.3}$$

with $n \in [1 : N]$ and $p \in [21 : 108]$. Figure 2.2 shows an example of a pitch feature.

2.3 Chroma Features

The pitch features introduced in the previous section capture the harmonic content of audio signals quite well. But in the context of the CONNECTOR they have a major drawback. They do not abstract away from the periodic pitch perception of the human auditory system. For a human, two tones for which one of them is a multiple in frequency of the other sound similar. We say they have the same "color" or *chroma*. In western music these chroma are denoted by $C, C^\sharp, D, D^\sharp, E, F, F^\sharp, G, G^\sharp, A, A^\sharp$ and B while adding a number label in case one wants to refer to a certain tone like $A1 = 440$ Hz and $A2 = 880$ Hz. Therefore two audio recordings that have a similar chroma progression should also have similar feature sequences. This is not always true for the pitch features.

The idea to overcome this problem is therefore to aggregate all pitches that belong to the same chroma class in a pitch feature vector. The result is a 12-dimensional feature vector called a *chroma vector* or *chroma feature*. To increase the robustness against differences in sound intensity or dynamics, the chroma vectors are then usually normalized by replacing every vector v by

$$v = \frac{v}{\|v\|_1} \tag{2.4}$$

where

$$\|v\|_1 = \sum_{i=1}^{12} |v(i)| \tag{2.5}$$

denotes the ℓ^1-norm of v. A sequence of chroma features then expresses the relative energy distribution of the underlaying signal within the twelve chroma bands. Chroma features

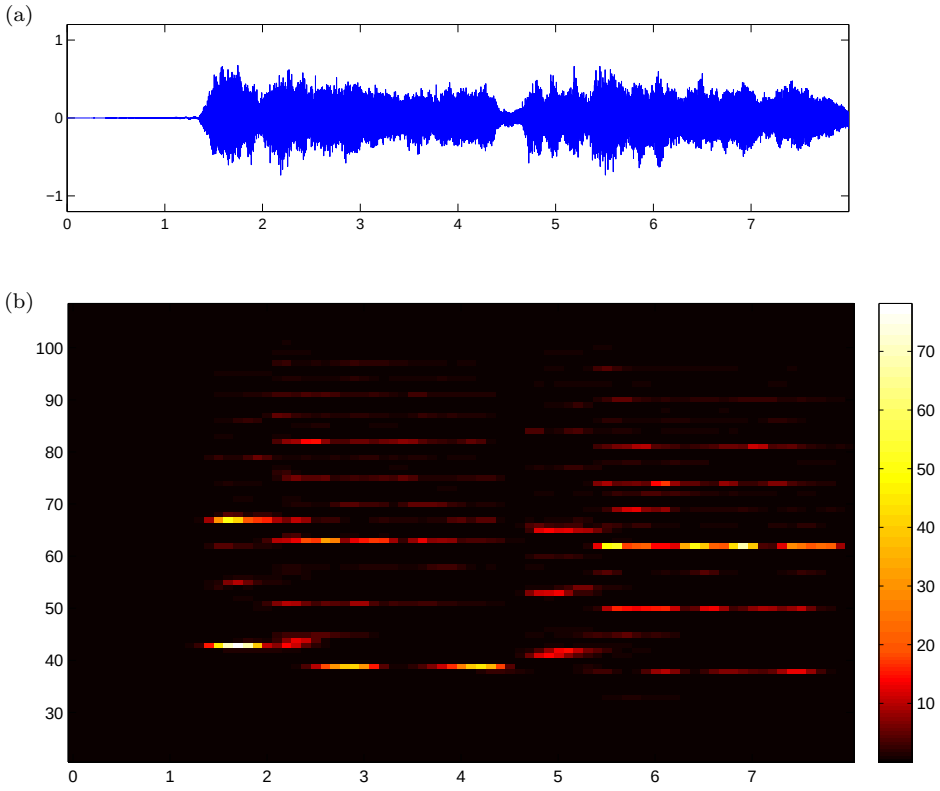

Figure 2.2. **(a)** The waveform of the first 8 seconds of Beethoven's 5^{th} Symphony in an orchestral version. Time is given in seconds. **(b)** The pitch feature computed from (a). The segments that were used to compute the STMSP have a length of 200 ms and overlap by 100 ms.

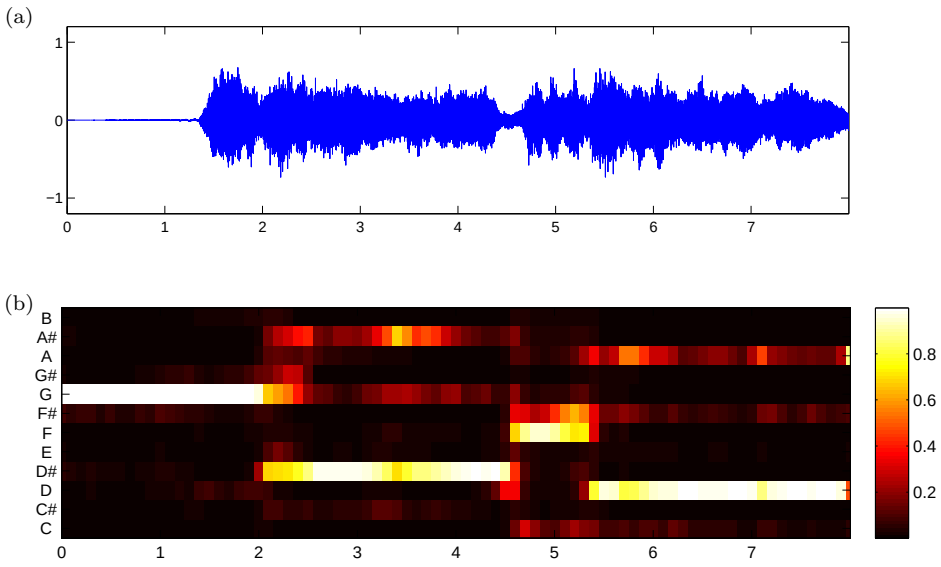

Figure 2.3. **(a)** The waveform of the first 8 seconds of Beethoven's 5^{th} Symphony in an orchestral version. Time is given in seconds. **(b)** The chromagram computed from the pitch feature visualized in Figure 2.2.

are usually visualized in a so called *chromagram*. See Figure 2.3 for an example.

2.4 Beat-Synchronous Chroma Features

The chroma features introduced in Section 2.3 still have a drawback. Since they are computed on segments of fixed length, the boundaries of those segments usually do not coincide with musically meaningful boundaries. *Time-adaptive* features are a solution to overcome this problem. The main idea of time-adaptive features is to abstract away from the actual timing information of the underlying audio signal and compute the features not on fixed length intervals but on musically meaningful segments [44]. By choosing those segments to be the intervals between two beat positions in an audio signal during the computation of the chroma features, we get *beat-synchronous chroma features*. See Figure 2.4 for an example.

Intuitively, the beat-synchronous chroma features are computed by computing the average of all fixed-length feature vectors within two beat positions of the pitch feature representation. This pitch feature is then transformed into a chroma feature in the standard way. For further details about the computation of beat-synchronous chroma features and time-adaptive features in general we refer to [44].

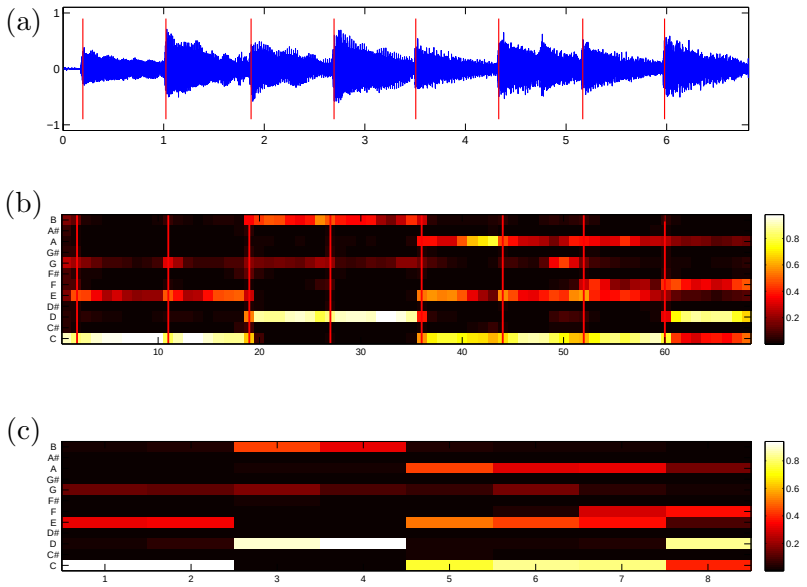

Figure 2.4. **(a)** The waveform of the first two measures of the song "let it be" from the Beatles. Time is given in seconds and the beat positions are indicated by red bars. **(b)** The chromagram of the audio signal from (a). The features were computed at a fixed rate of 10 Hz. Again the beat positions are indicated by red bars. Note that the note boundaries are blurred due to the fixed length segmentation. **(c)** The beat-synchronous chromagram computed from the signal from (a) and the beat positions.

Chapter 3

Time-Scale Modification

Time-scale modification is the task of altering the time-scale of audio signals, that means making the tempo of audio recordings faster or slower. In the field of digital signal processing there exists a large variety of different algorithms that are designed to fulfill this task. These algorithms are typically referred to as time-scale modification algorithms, or abbreviated by TSM algorithms. While we discuss existing algorithms in Chapters 4, 5 and 6 we will address the topic of time-scale modification of DT-audio signals in general in this chapter. In Section 3.1 we give an introduction to the field of time-scaling of audio signals. Related work is introduced in Section 3.2 and finally basic definitions as well as technical remarks are given in 3.3.

3.1 Introduction

The task of manipulating the time-scale of an audio signal is a frequently occurring one. For example in the field of DJing it is often important to have multiple audio clips which share the same tempo. A DJ then creates whole songs by overlaying and sequencing those clips. For example he could use a drum clip, a bass clip, a melody clip and some effect clips, all played at the same time. But if the drums, the bass and the melody are not on the same speed level, the result will sound chaotic. He therefore has to take care that the tempos of all simultaneously played clips are the same. But most often audio clips are not available in different versions for different tempos. The solution is to manipulate the time-scale of all clips such that they are all on the same speed level.

Another scenario are commercials where an advertising text needs to fit into a predefined time frame of mostly 30 or 60 seconds. Since it is extremely difficult for a voice actor to meet this constraint exactly, the text is most commonly recorded and the speed of the audio recording is adjusted afterwards such that the length of the recording matches the given time. Furthermore, research has shown that the human brain works most efficiently when listening to spoken text if the speed of the speaker is about 200 - 300 words per minute [15]. This is the average reading speed of an adult. Unfortunately, the average rate of speech is in the neighborhood of 100 - 150 words per minute. This fact is used frequently in commercials since by speeding up the recorded advertising text one is able

to give more information in less time and the faster text is even easier to process by the human brain.

These two applications are instances of problems where a simple *linear time-scale modification* suffices. For this purpose we can see a TSM algorithm as a method that receives an audio signal along with some descriptor which describes the intended time-scale modification, in this case a constant *time-stretch factor*. The time-stretch factor determines how the time axis of the given audio should be rescaled for the output. For example a time-stretch factor of 2 should yield an audio signal that is twice the length of the audio signal given to the algorithm. The straightforward approach to realize such a time-scale modification of an audio signal is to either stretch or compress the waveform of the given audio signal, depending on the time-stretch factor. This is done by resampling the given signal but still interpreting it as a signal sampled at the old sampling rate. Algorithm 1 shows how the resampling of a signal can be computed. Unfortunately, this simple approach yields unacceptable results for most applications. When resampling the signal one does not only change the time-scale, but also the pitch-scale of the signal at the same rate (see Figure 3.1). The effect is the same as when playing a record or a tape recording at a higher or lower speed than it is intended to be played. This might be acceptable for some kinds of audio signals with non-harmonic content, like for example drum beats, where slight changes in the pitch-scale are not directly audible as such. But, for example, for audio signals containing human voice this leads to the so called *chipmunk effect*[1] which makes the human voice sound completely unnatural and alienated by heightening or lowering its pitch. In the context of the CONNECTOR, the resampling approach is even not applicable at all since we want to manipulate the time-scale of audio signals that are similar in the harmonic sense. A pitch shift destroys this similarity completely.

Algorithm 1: Resampling

Data: DT signal x sampled at a sampling rate f_s, time-stretch factor t.
Result: DT signal y that is a resampled version of x.
begin
  $z \longleftarrow$ interpolate x to a CT signal;
  $y \longleftarrow$ sample z at a rate of $f_s \cdot t$;

We therefore need TSM algorithms that are capable of changing the time-scale of an audio signal without altering its pitch-scale. In the optimal case these algorithms should produce an audio signal that sounds as if the content of the signal was produced on a different time-scale. For example for a musical performance the output should sound as if all the musicians just played at a different tempo.

Historically, the next step towards this goal was to invent the so called *Variable Speech Compression* [15]. Having an audio signal, this technique just takes small segments from the signal at equidistant positions and either removes them or doubles them. Depending on the length of those segments and whether they are removed or doubled, the audio signal is more or less shortened or lengthened. This technique was for example implemented in cassette recorders. Here, the tape was first played on a higher or lower speed, leading

[1] `http://en.wikipedia.org/wiki/Alvin_and_the_Chipmunks`, last consulted in October 2011

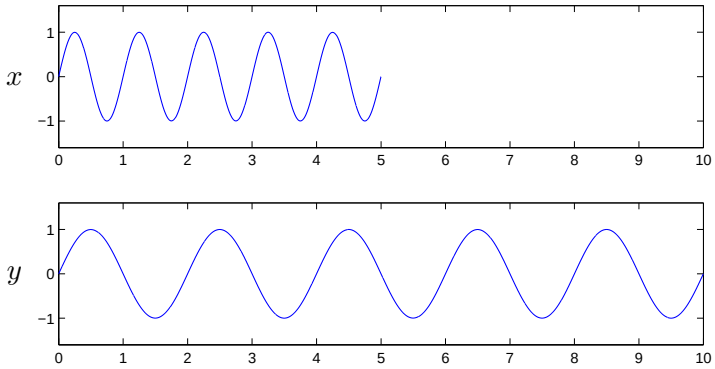

Figure 3.1. The sinusoidal signal x is sampled at a rate of 22050 Hz and has a frequency of 1 Hz (the time on the x-axis is given in seconds). By resampling the signal to 44100 Hz we get the signal y. The length of y is double the length of the signal x, but at the same time also the wavelength of x is doubled. Therefore the frequency of the sinusoidal signal y is 0.5 Hz. It is not possible to change only the duration of a signal without changing the pitch and vice versa by purely resampling the signal.

to time-scaled, but pitched signals. Then small segments were recorded from the tape, resampled to pitch them back to their initial pitch and then fed to the output, while either skipping certain parts of the tape when increasing the speed or doubling the resampled segments when slowing the audio down. The quality of time-scaled audio signals produced by variable speech compression was rather good, especially when manipulating the time-scale of speech recordings. But stuttering artifacts were always audible and the approach was therefore far away from the goal of artifact free time-scale modification. Nevertheless, this technique has a lot in common with one of the first digital TSM algorithms, the OLA (*Overlap and Add*) technique which we discuss in Chapter 4.

An interesting fact is that *pitch-scale modification*, which is the task of changing only the pitch of an audio signal while the length of the signal is preserved, is actually the same as time-scale modification. While at the first glance these problems seem unrelated, solving one of them also solves the other. Assume one has a perfect TSM algorithm that is capable of time-scaling any audio signal without introducing artifacts, but the task is to lower the pitch of an audio signal without altering the length. One starts by using the TSM algorithm to shorten the duration of the audio signal without altering its pitch. Afterwards, one uses the naive resampling approach to stretch the shortened signal to its initial length again and at the same time lower its pitch. This results in an audio signal, that has the same length as the initial signal but that is lowered on the pitch-scale. The heightening of the pitch-scale works analogously.

3.2 Related Work

A lot of work has been done and is still ongoing in the field of time-scale modification of audio signals. The Variable Speech Compression [15] that was briefly introduced in Section 3.1 was one of the first contributions to this field. For modern TSM algorithms there exist in principle two main approaches: They either work in the time-domain or in

the frequency-domain.

Algorithms working in the time-domain try, similarly to the Variable Speech Compression, to take small segments from specific positions in the input audio signal and resynthesize them to a new, time-scale modified version of the input audio signal. Most of these algorithms follow a basic scheme, the so called OLA (Overlap and Add) technique. This technique is presented in [36] and is also discussed in detail in Chapter 4. Many high quality TSM algorithms originate from this technique. Examples are the WSOLA algorithm (Wave Similarity Overlap and Add) [43], which is also discussed in Chapter 4, the SOLA algorithm (Synchronized Overlap and Add) [37] and the PSOLA algorithm (Pitch-Synchronous Overlap and Add)[32]. These algorithms often suffer from stuttering artifacts at transient regions in audio signals. Therefore the quality of the results can be improved by giving these transient regions a specialized treatment. An approach to improve the quality of WSOLA can be found in [19] and our own approach is discussed in Chapter 6.

The other family of TSM algorithms works in the frequency-domain. These algorithms approach the problem by constructing frequency spectra that correspond to a time-scaled version of the input audio signal. By transforming these spectra back to the time-domain they produce the output audio signal. The most prominent algorithm of this family is the Phase Vocoder [14, 1, 9, 10, 42], which is discussed in detail in Chapter 5.

There exist also some hybrid approaches that work partially in the time-, and partially in the frequency domain. Such algorithms are presented in [25, 18, 11]. A recent and very intersting approach which combines WSOLA and the Phase Vocoder can be found in [31].

3.3 General Definitions and Remarks

In the context of the CONNECTOR, as well as in many other applications it is not sufficient to have a TSM algorithm that only allows for a linear time-scale modification. Often it is necessary to scale the time of an audio signal in a non-linear fashion. For example in the CONNECTOR we want to synchronize the beats of two audio signals. But those beat positions do not need to be equidistant. In fact, this is not the case in most musical pieces performed by humans. Not only that unskilled musicians tend to miss the correct beat position, but also professional musicians often play notes a little bit before or after the beat (pushing/lay back) to create certain moods in their music. This is referred to as *tempo rubato* [23]. Furthermore agogics like ritardando or accelerando (get slower/get faster) make musicians change the speed of the played piece and therefore the temporal distance between beats. In such cases a non-linear time-scale modification is necessary to synchronize the beat positions of two audio recordings.

Modern TSM algorithms are normally capable of handling non-linear time-scale modifications. To this end they get a *time-stretch function* τ along with the input signal as a descriptor of the intended time-scale manipulation. Intuitively a signal f and a signal g which is a time-scale modified version of f with respect to the time-stretch function τ should then satisfy the formula

$$\forall t \in [0, T) : g(\tau(t)) = f(t) \quad , \tag{3.1}$$

where T is the duration of f. This formula states that every single point in time of signal f is mapped to a point in time in g by the time-stretch function τ. Unfortunately the formula does not specify what we intended. Figure 3.2 shows an example of a signal, a time-stretch function and a time-scaled version of the signal which satisfy the formula. The problem is that in an audio signal, information about time, pitch, timbre and many other aspects of sound are mingled together. Formula 3.1 specifies a time-scale manipulation that manipulates all properties of a signal at once which leads to the same artifacts as when speeding up or slowing down the spin of a record according to the time-stretch function while playing it, or, in the digital world, the resampling approach discussed in Section 3.1. But our goal is to manipulate only the time axis of an audio signal, without changing any other properties. We therefore have two seemingly contradicting goals. On the one hand we want to distort the audio signal temporally on a global scale to achieve the intended time-scale modification, but on the other hand we want to preserve it locally to maintain aspects like pitch and timbre. To define this property and therefore to overcome the shortcomings of Equation 3.1 we need a better way to talk about corresponding points in time in two signals. We are looking for a relation that allows us to compare the local content of two signals at given points in time. Defining such a relation is difficult since it is not even easy to explain what exactly a point in time in an audio signal is. When we break it down to the waveform, it is just the displacement in the air pressure at the given time, measured and quantized in the amplitude of the recorded signal. But this information is not enough to preserve aspects like pitch and timbre of the signal when time-scale modifying it. The problem is that not a single point in time makes the difference in air pressure that the human ear perceives as sound, but a sequence of different air pressure levels does. It is therefore necessary that the relation we are looking for does not connect pure points in time, but short time segments.

We now first give the definition of the term time-stretch function.

Definition 3.1 *A* Time-Stretch Function τ *for a CT signal* f *is a strictly monotonously increasing function* $\tau : [0, T) \to \mathbb{R}$ *where* T *is the duration of* f*. The domain of* τ *represents the time-axis of* f *and the range the time-axis of a time-scale modified version of* f*.*

To be able to give at least an intuitive definition of what it means that an audio signal f is time-scale modified with respect to a time-stretch function τ we introduce a relation $\approx$ and give it the vague meaning of *"similar to"*. Note that finding an operational definition of this relation that describes the desired property perfectly is the same as solving the TSM problem artifact free since we could just use the definition to compute the output signal. Therefore building a TSM algorithm is somehow similar to approach the problem of defining this relation.

Definition 3.2 *A* CT window function w of size ℓ *is a function* $w : \mathbb{R} \to \mathbb{R}$ *such that* $w(t) > 0$ *for all* $t \in T = [-\frac{\ell}{2}, \frac{\ell}{2})$ *and* $w(t) = 0$ *for* $t \in \mathbb{R} \backslash T$*.*

Definition 3.3 *A CT signal* g *is a* time-scale modified version of a CT signal f with respect to τ *if*

$$\forall t \in [0, T) : g(\tau(t) + s) \cdot w(s) \approx f(t + s) \cdot w(s) \quad ,$$

where T *is the duration of the signal* f*,* w *is a window function of window size* ℓ *and* $s \in (-\frac{\ell}{2}, \frac{\ell}{2})$*.*

Note that in the above definition the relation $\approx$ relates windowed segments of the two signals and not single points in time.

Since we want to argue mainly about digital signals we need to adapt the definition to DT signals.

Definition 3.4 *Let $x : [1 : N] \to \mathbb{R}$ be a DT signal and τ a time-stretch function for a continous version of x. A* Discretized Time-Stretch Function *$\hat{\tau}$ of τ for x is a strictly monotonously increasing function $\hat{\tau} : A \to [1 : M]$ where $A \subseteq [1 : N]$ $1, N \in A$ and M is the length of the time-scale modified signal and it holds that*

$$\begin{aligned} &(i) \quad \hat{\tau}(1) = 1 \\ &(ii) \quad \hat{\tau}(N) = M \\ &(iii) \quad \forall n \in A \backslash \{1, N\} : \hat{\tau}(n) = [\tau([n]_{time})]_{sample} \end{aligned}$$

The functions used in (iii) are defined as $[n]_{time} = n \cdot p_s$ and $[t]_{sample} = round(\frac{t}{p_s})$ where p_s is the sampling period of x.

A discretized time-stretch function $\hat{\tau}$ does not need to be defined for all sample positions of a DT signal x. Nevertheless this is often necessary. We therefore introduce the interpolated time-stretch function τ'.

Definition 3.5 *An* Interpolated Time-Stretch Function *τ' of a discretized time-stretch function $\hat{\tau}$ for a DT signal x is a monotonously increasing function $\tau' : [1 : N] \to [1 : M]$ where N is the length of x and M is the length of the time-scale modified version of x. The function τ' is computed from $\hat{\tau}$ by discrete linear interpolation.*

When defining TSM algorithms it is also often necessary to invert the time-stretch function. In the discrete world we therefore have to overcome the problem that the interpolated time-stretch function τ' is not necessarily strictly monotonously increasing. We define the inverse of an interpolated time-stretch function as follows.

Definition 3.6 *An* Inverted Interpolated Time-Stretch Function *${\tau'}^{-1}$ of an interpolated time-stretch function τ' for a DT signal x is a monotonously increasing function ${\tau'}^{-1} : [1 : M] \to [1 : N]$ where N is the length of x and M is the length of the time-scale modified version of x. The function ${\tau'}^{-1}$ is computed from $\hat{\tau}^{-1}$ by discrete linear interpolation.*

Finally we adapt Definition 3.3 to the discrete world.

Definition 3.7 *A* DT window function *w of size L is a function $w : \mathbb{Z} \to \mathbb{R}$ such that $w(m) > 0$ for all $m \in M = [-\lfloor \frac{L-1}{2} \rfloor : \lfloor \frac{L}{2} \rfloor]$ and $w(m) = 0$ for $m \in \mathbb{Z} \backslash M$.*

Definition 3.8 *A DT signal y is a* time-scale modified version of a DT signal x with respect to τ' *if*

$$\forall n \in [1 : length(x)] : y(\tau'(n) + m) \cdot w(m) \approx x(n + m) \cdot w(m) \quad ,$$

where w is a window function of size L and $m \in [-\lfloor \frac{L-1}{2} \rfloor : \lfloor \frac{L}{2} \rfloor]$.

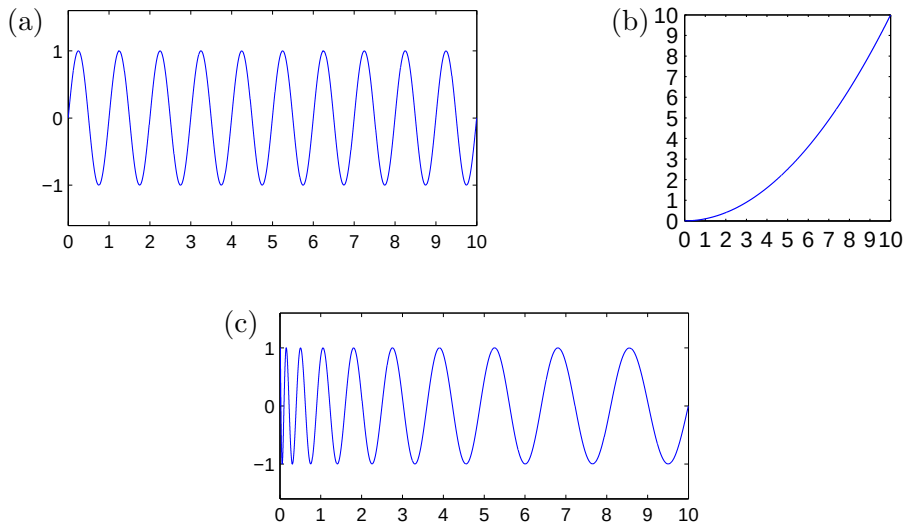

Figure 3.2. **(a)** A sinusoidal signal with frequency 1 Hz and a duration of 10 seconds. **(b)** The non-linear time-stretch function τ. **(c)** The time-scale modified resulting signal that satisfies Formula 3.1. Note that this signal is not the intended result since the frequency of the sinusoid changes over time and therefore the pitch of the signal changes as well. Note further that the intended result is exactly **(a)** again. This is because our input signal is essentially one tone lasting 10 seconds. The time-stretch function τ suggests that the output signal should also have a length of 10 seconds. Since we do not want to alter the pitch of the signal in any way the intended output is exactly the input again.

It will always be clear from the context wether we mean the continuous time-stretch function τ, the discretized version $\hat{\tau}$ or the interpolated time-stretch function τ' and will therefore only use the symbol τ for the sake of simplicity.

Chapter 4

WSOLA

WSOLA (Wave Similarity Overlap and Add) is a time-domain TSM algorithm from the family of OLA algorithms and was first introduced in [43]. In this section we discuss it in detail. We first introduce the basic OLA technique in Section 4.1 and afterwards the improvements that were done in WSOLA in Section 4.2. Finally we briefly have a look at occurring artifacts in Section 4.3.

4.1 OLA

The basic OLA technique is very simple and follows roughly the ideas of the Variable Speech Compression. The main idea of the algorithm is to cut out small segments from the input audio and concatenate them by crossfading from one segment to the next to achieve the desired time-scale modification. To be able to describe the technique properly we first need to introduce some definitions and remarks.

Remark 4.1 *We will use window functions massively throughout this chapter. For the sake of convenience we will therefore denote the size of a window function w by w_ℓ.*

Definition 4.1 *For x being a DT signal of length N and w being a window function of size N, we define*

$$x \cdot w$$

to be the pointwise multiplication of x and w.

Definition 4.2 *A* windowed audio segment z from an DT signal x using the window function w around window position p *is defined as*

$$z = x[b_w^p : e_w^p] \cdot w \quad ,$$

where $b_w^p = p - \lfloor \frac{w_\ell - 1}{2} \rfloor$ is the beginning of the windowed audio segment z in x and $e_w^p = p + \lfloor \frac{w_\ell}{2} \rfloor$ is the end of the windowed audio segment z in x.

Definition 4.3 *Let w be a window function. In a sequence of windows in which all windows can be described by a shifted version of w and in which all windows are overlapping by a constant factor o we call the distance between the centers of two adjacent windows the* standard window offset η_o^w *and compute it as*

$$\eta_o^w = (1 - o) \cdot w_\ell \quad .$$

Given to the OLA algorithm are an input signal x, an overlap factor o, a window function w and a time-stretch function τ. The goal of the algorithm is to produce an audio signal y that is a time-scale modified version of x with respect to τ. The OLA technique realizes this task by copying audio segments that are windowed using w from the input signal x to the output signal y. It is important that the size of the used window function w is longer than one pitch period of the lowest fundamental frequency contained in the input signal x, since a windowed audio segment from x using w should contain harmonically meaningful content. In the output these windowed audio segments are overlapped by the constant overlap factor o and added up, which yields a crossfade from one segment to the next. This procedure is the origin of the name of the technique (Overlap and Add).

The first step of the OLA technique is to generate a vector of output window positions γ. This vector specifies positions where the windowed audio segments from the input will be placed in the output. It is fully determined by the length of the output signal y (which by itself is given by the time-stretch function τ), the length of the window function w_ℓ and the overlap factor o. This is because it specifies the positions of a sequence of windows that are all overlapping by the same factor. Therefore all entries in γ are equidistant and depend only on w_ℓ and o. The length of γ is determined by the length of y since no window position in γ may exceed the length of y.

Now we can compute γ by

$$\begin{array}{ll} (i) & length(\gamma) = \lceil \frac{length(y)}{\eta_o^w} \rceil \\ (ii) & \gamma(1) = 1 \\ (iii) & \gamma(n) = \gamma(n-1) + \eta_o^w \text{ for } n \in [2 : length(\gamma)] \end{array} \tag{4.1}$$

One can now imagine the output signal y as a sequence of slots that need to be filled with windowed audio segments from the input. The positions of these slots are all equidistant and given by γ. To fill these slots with windowed audio segments such that the output signal y is a time-scale modified version of x with respect to the time-stretch function τ, we construct a vector of input window positions σ which is dependent on τ. It is computed by

$$\sigma(n) = \tau^{-1}(\gamma(n)) \tag{4.2}$$

for all $n \in [1 : length(\gamma)]$. Every slot in y specified by $\gamma(n)$ is then filled with the windowed audio segment at window position $\sigma(n)$. Figure 4.1 gives a visual example of how the OLA technique works.

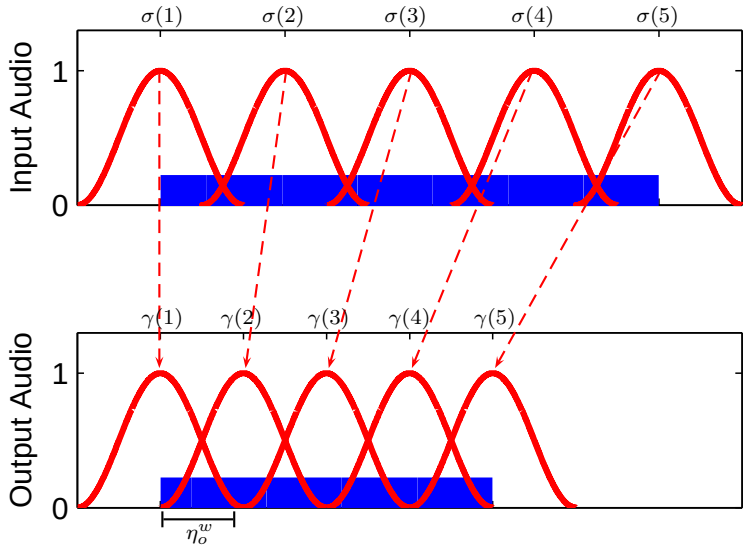

Figure 4.1. Schematic mechanics of OLA. Signals are symbolized by blue bars. The input signal is windowed at specified positions σ. The resulting windowed segments are then overlapped by a constant factor o (it holds that $\gamma(n+1) - \gamma(n)$ is constant, in this case the overlap factor o is 0.5 and the standard window offset η_o^w therefore is half the window size), added up and copied to the output. The example shows how OLA produces a version of the input signal that is speeded up by a constant factor (the windows in the input are sampled equidistantly along the input signal and $\sigma(n+1) - \sigma(n) > \gamma(n+1) - \gamma(n)$).

Mathematically we can describe the synthesis of y as

$$y(n) = \frac{\sum_{k=1}^{length(\sigma)} w(n - \gamma(k)) \cdot x(n - \gamma(k) + \sigma(k))}{\sum_{k=1}^{length(\sigma)} w(n - \gamma(k))} \quad . \tag{4.3}$$

Intuitively Equation 4.3 simply computes the sum of all windowed audio segments from the input signal x that contribute to sample position n in the output signal y. Note that the denominator is used to normalize the output y at position n in case the overlapping windows at position n in the output do not add up to 1. This normalization is important to avoid amplitude modulations in the output signal y (see Figure 4.2 for an example). In OLA techniques, the overlap factor o is typically set to 0.5 and the window function w usually describes a Hann-window. With these settings the denominator of Equation 4.3 is always 1.

From Equation 4.3 we can derive Algorithm 2. Note that the very last step of the algorithm applies the discussed normalization to the output signal y and is obsolete in case $o = 0.5$ and w being a Hann-window.

By extracting and rearranging small segments with harmonically meaningful content from the input audio x, the OLA technique tries to give the relation $\approx$ that we introduced in Chapter 3 the meaning of "harmonically similar to". The Short Time Fourier Transform

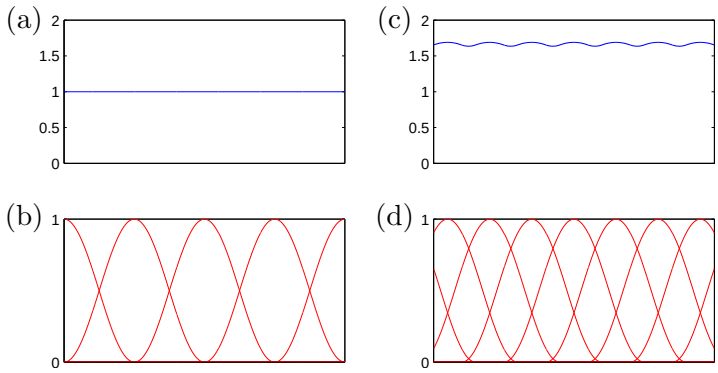

Figure 4.2. (a) and (b) The Hann-windows in (b) are overlapped by an overlap factor of $o = 0.5$. The plot in (a) shows the pointwise addition of all windows. The summed values are constantly equal to 1. **(c) and (d)** The Hann-windows in (d) are overlapped by an overlap factor of $o = \frac{7}{10}$. The plot in (c) shows the pointwise addition of all windows. One can observe that summed values are not constant. Note that the function plots shown in (a) and (c) describe exactly the denominator of Equation 4.3 if we assume that w is the used Hann-window function and the window positions resulting from the overlap factors are given in a vector γ.

Algorithm 2: OLA

Data: DT signal x, window function w, overlap factor o, time-stretch function τ.
Result: DT y that is a time-scale modified version of x.
begin

`/* Compute` γ `*/`
$\gamma(1) = 1$;
for $i \leftarrow 2$ **to** $\lceil \frac{length(y)}{\eta_o^w} \rceil$ **do**
  $\gamma(i) \longleftarrow \gamma(i-1) + \eta_o^w$;

`/* Compute` σ `*/`
for $i \leftarrow 1$ **to** $length(\gamma)$ **do**
  $\sigma(i) \longleftarrow \tau^{-1}(\gamma(i))$;

`/* Overlap and Add */`
for $j \leftarrow 1$ **to** $length(\sigma)$ **do**
  $frame \longleftarrow x[b_w^{\sigma(j)} : e_w^{\sigma(j)}] \cdot w$;
  $y[b_w^{\gamma(j)} : e_w^{\gamma(j)}] \longleftarrow y[b_w^{\gamma(j)} : e_w^{\gamma(j)}] + frame$;

`/* Adjust possible amplitude modulations */`
$y \longleftarrow$ adjustAmplitude(y, w, o)

[26] is a tool to analyze the frequency content of a DT signal. It is defined by

$$X(t,k) = \sum_{n\in\mathbb{Z}} x(t+n)\cdot w(n)\cdot e^{-\frac{2\cdot\pi\cdot i\cdot k\cdot n}{N}} \quad , \tag{4.4}$$

where N is the size of the discrete Fourier Transform, w is a window function of size N, t is the time given in samples and k is the index of the frequency bin with center frequency $\frac{2\cdot\pi\cdot k}{N}$. Let X_ν be defined as

$$X_\nu = (X(\nu(1),\cdot), X(\nu(2),\cdot), ..., X(\nu(Z),\cdot)) \quad , \tag{4.5}$$

where ν is some vector of sample positions of length Z. We can therefore describe the instantiation of $\approx$ that OLA is designed to fulfill as

$$\begin{gathered}\forall n \in [1 : length(x)] : y(\tau(n)+m)\cdot w(m) \approx x(n+m)\cdot w(m) :\Leftrightarrow \\ Y_\nu \approx_{LS} X_{\tau^{-1}(\nu)}\end{gathered} \tag{4.6}$$

where $\approx_{LS}$ means "close to" in the least-squares sense, w is a window function of size L and $m \in [-\lfloor\frac{L-1}{2}\rfloor : \lfloor\frac{L}{2}\rfloor]$. Equation 4.6 defines that the harmonic content of the input and the output signal have to be similar at a given set of pairs of points in time

$$\left(\left(\nu(1),\tau^{-1}(\nu(1))\right),\left(\nu(2),\tau^{-1}(\nu(2))\right),...,\left(\nu(Z),\tau^{-1}(\nu(Z))\right)\right) \quad , \tag{4.7}$$

where the first entry of a pair specifies a point in time in the output signal and the second entry a point in time in the input signal. If we instantiate ν with γ, OLA fulfills this definition by design since the window positions in OLA are chosen exactly this way (recall that we calculated σ as $\sigma(n) = \tau^{-1}(\gamma(n))$.

The quality of audio signals created with the OLA technique is acceptable as long as the content of the signal is of a non-harmonic nature like for example speech. But when it comes to the time-scaling of signals with harmonic content, like for example music recordings, the technique introduces a lot of modulation artifacts. These artifacts result from cancellation effects that occur when the phases of the fundamental frequencies of two overlapping windows in the output do not match. This also introduces most certainly phase jumps (see Figure 4.3) which then lead to a dissonant sound of the output signal. This is of course not acceptable in most scenarios where music should be time-scale manipulated. The reason for the occurring artifacts is that the instantiation of $\approx$ described in Equation 4.6 which OLA fulfills is not sensitive to the phase relationships existing in the the input signal x. It is therefore necessary to revise this definition.

4.2 Improvements to OLA - The WSOLA Algorithm

The problem with basic OLA is that the technique is not sensitive to the input signal itself. It just copies windowed audio segments from fixed positions in the input audio signal to fixed positions in the output audio signal. The underlying signal has no influence on the

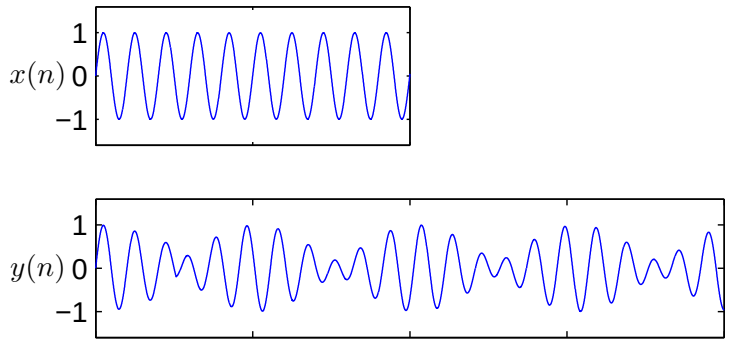

Figure 4.3. The input signal x is a simple sinusoid. The signal is stretched using the OLA technique by a factor of 2. OLA is not capable of maintaining the structure of x and introduces modulation and phase jump artifacts into the output signal y.

algorithm at all. Admittedly the frequency content is maintained locally, but so are the phases of the copied segments. When overlapping those segments, artifacts are introduced in case that the phases do not match. The main idea of high quality TSM algorithms is therefore to keep only the magnitude spectra similar locally while taking care that no new phase jumps are introduced (optimally phase jumps that existed in the input signal x should persist in the output signal y) instead of maintaining the similarity of the full Fourier spectra (recall that a Fourier spectrum can be divided into a magnitude spectrum and a phase spectrum since a Fourier spectrum is a vector of complex numbers). We can therefore define a new instantiation of $\approx$ which most modern TSM algorithms aim to fulfill.

$$\begin{aligned} \forall n \in [1 : \mathit{length}(x)] : y(\tau(n) + m) \cdot w(m) \approx x(n + m) \cdot w(m) :\Leftrightarrow \\ (i) \quad |Y_\nu| \approx_{LS} |X_{\tau^{-1}(\nu)}| \\ (ii) \quad \text{No new phase jumps are introduced in } y \end{aligned} \tag{4.8}$$

where ν is some vector of samplepositions in y, w is a window function of size L and $m \in [-\lfloor \frac{L-1}{2} \rfloor : \lfloor \frac{L}{2} \rfloor]$.

In OLA, the artifacts mainly result from discontinuities in the phase of the fundamental frequency of the signal. The basic idea to overcome this issue in most time-domain TSM algorithms is to give the algorithm some flexibility when it comes to the arrangement of the windows such that the fundamental frequency is maintained as well as possible. WSOLA therefore introduces a position tolerance Δ_{max} for the input window positions specified by σ and therefore relaxes the time-stretch function τ. But even though WSOLA therefore deviates from the given time-stretch function τ, it yields results of much better quality since audio artifacts are reduced significantly and the tiny deviations from τ can be neglected in practice. For an example compare Figure 4.3 to Figure 4.4.

We define our new input window position vector as

$$\sigma_{relaxed}(k) = \sigma(k) + \Delta_k \quad , \tag{4.9}$$

where $k \in [1 : \mathit{length}(\sigma)]$ and $\Delta_k \in [-\Delta_{max} : \Delta_{max}]$. The WSOLA synthesis equation

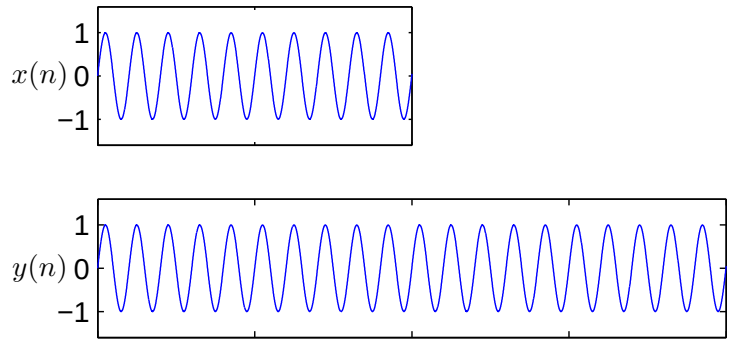

Figure 4.4. The same input signal as in Figure 4.3 is stretched by a factor of 2 using the WSOLA technique. WSOLA is able to maintain the sinusoidal structure of the input signal.

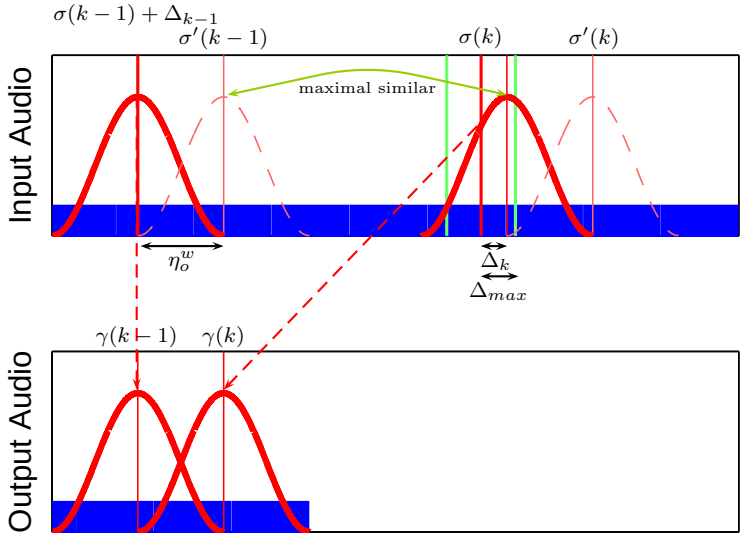

Figure 4.5. One iteration of the WSOLA algorithm. Like in Figure 4.1 signals are symbolized by blue bars. The windowed audio segment around $\sigma(k-1)$ is copied to the output. WSOLA now finds an offset $\Delta_k \in [-\Delta_{max} : \Delta_{max}]$ (the borders of the tolerance region are indicated by green lines) for the next window position $\sigma(k)$ such that the windowed audio segment around $\sigma(k)+\Delta_k$ is maximal similar to the windowed audio segment around the natural progression $\sigma'(k-1)$. After having found the offset Δ_k the windowed audio segment around $\sigma(k)+\Delta_k$ is copied to the output and the process is repeated for the next input window position. Note that for the sake of clarity the input window positions are spaced very far apart in contrast to Figure 4.1.

therefore becomes

$$y(n) = \frac{\sum_{k=1}^{length(\sigma)} w(n-\gamma(k)) \cdot x(n-\gamma(k)+\sigma(k)+\Delta_k)}{\sum_{k=1}^{length(\sigma)} w(n-\gamma(k))} \quad . \tag{4.10}$$

It remains to show how the Δ_k are found. Figure 4.5 visualizes the process while it is explained in detail in the following. Remember that we want to choose the input window positions such that the phase of the fundamental frequency of the input signal is preserved as good as possible. In the optimal case we find an offset Δ_k such that when overlapping the windowed audio segment around $\sigma(k) + \Delta_k$ with the windowed audio segment at $\sigma(k-1) + \Delta_{k-1}$ we have no cancellation effects at all. Unfortunately, this is in general only the case if

$$\sigma(k) + \Delta_k = \sigma(k-1) + \Delta_{k-1} + \eta_o^w \quad . \tag{4.11}$$

In this scenario the distance between the two input window positions is the standard window offset η_o^w. This is by design of WSOLA also the distance between the centers of overlapping windowed audio segments in the output. The overlapping parts of the two windowed audio segments in the output therefore have exactly the same phase since they origin from exactly the same segment in the input signal. We therefore call the sample position $\sigma(k-1) + \Delta_{k-1} + \eta_o^w$ the *natural progression* $\sigma'(k-1)$ of the sample position $\sigma(k-1) + \Delta_{k-1}$ and generally define it by

$$\sigma'(k) = \sigma(k) + \Delta_k + \eta_o^w \quad . \tag{4.12}$$

Note that in case Equation 4.11 holds for all $k \in [1 : length(\sigma)]$ this indicates a time-stretch function that is the identity and therefore a global time-stretch factor of 1. In case the time-stretch function τ is not the identity, the natural progression of $\sigma(k-1) + \Delta_{k-1}$ will most probably not lie in the tolerance region $[\sigma(k) - \Delta_{max} : \sigma(k) + \Delta_{max}]$. Therefore we are looking for an offset $\Delta_k \in [-\Delta_{max} : \Delta_{max}]$ such that the windowed audio segment around $\sigma(k) + \Delta_k$ is as similar as possible to the windowed audio segment around the natural progression of $\sigma(k-1) + \Delta_{k-1}$. Note that here the term *similar* indeed means pointwise similarity. This offset can therefore be found efficiently by using the *cross-correlation measure*.

The cross-correlation for two DT signals x and y is defined by

$$(x * y)(n) = \sum_{m=-\infty}^{\infty} \bar{x}(m) \cdot y(n+m) \quad , \tag{4.13}$$

where $\bar{x}$ is the complex conjugate of x. Intuitively the measure shifts the signal y by n samples and computes the sum of the product of x and the shifted y. If a shifted version of y is similar to x, then positive and negative amplitudes of the two signals are roughly aligned which yields high values for the product of the two signals and therefore a high value of the sum. In contrary, if x is not similar to the shifted version of y, positive and negative amplitudes will cancel out to a high extend and therefore leave the sum of the products small. A peak in the cross-correlation at position n therefore indicates that when shifting the signal y by $-n$, the overlapping parts of the signals x and y can be considered similar. In the context of WSOLA we can apply this measure as follows. Having the last input window position fixed at $\sigma(k-1) + \Delta_{k-1}$ (Δ_1 is always 0, therefore there always exists a k such that the $(k-1)^{th}$ window position is already fixed) we are now looking for the Δ_k such that the windowed audio segments around $\sigma'(k-1)$ and $\sigma(k) + \Delta_k$ are as similar as possible. In other words we are looking for an offset $\Delta_k \in [-\Delta_{max} : \Delta_{max}]$ such that $x[b_w^{\sigma'(k-1)} : e_w^{\sigma'(k-1)}]$ is as similar as possible to a subsegment of length w_ℓ in

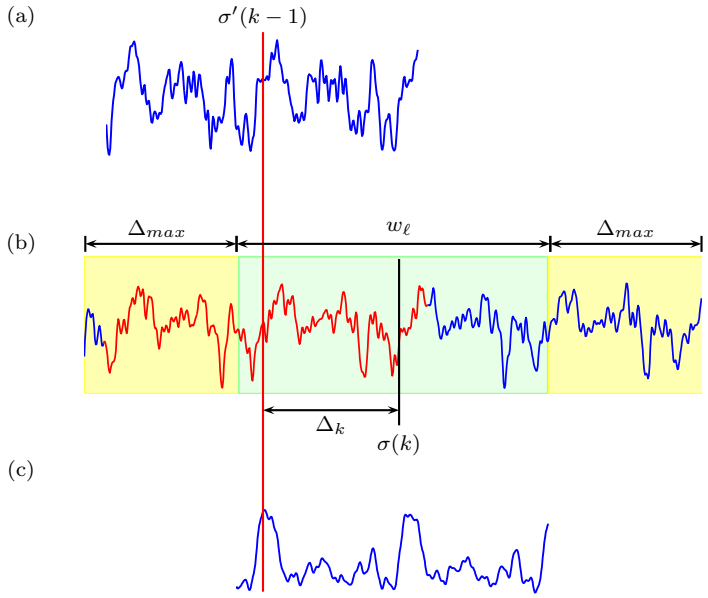

Figure 4.6. (a) The audio segment around the natural progression of $\sigma(k-1)+\Delta_{k-1}$. We can write it as $x[b_w^{\sigma'(k-1)} : e_w^{\sigma'(k-1)}]$. The position $\sigma(k-1)+\Delta_{k-1}$ is the last fixed input window position. **(b)** The segment $x[b_w^{\sigma(k)} - \Delta_{max} : e_w^{\sigma(k)} + \Delta_{max}]$. In this segment we are looking for a subsegment of length w_ℓ that is as similar to the segment from (a) as possible. This subsegment, found using the cross-correlation from (c), is marked in red.**(c)** The cross-correlation of the two segments. Picking the highest peak from the cross-correlation yields the offset of the segment from (a) to the segment from (b) such that they are as similar as possible. From this offset we can compute Δ_k.

$x[b_w^{\sigma(k)} - \Delta_{max} : e_w^{\sigma(k)} + \Delta_{max}]$. We now can simply compute the cross-correlation of those two audio segments. Picking the maximizing index of the cross-correlation yields an offset, and therefore an Δ_k, that yields the intended similarity. Figure 4.6 visualizes this process.

Note that the cross-correlation is similar in nature to the convolution and can therefore be computed efficiently in the frequency domain exploiting that

$$\widehat{x * y} = \overline{\widehat{x}} \cdot \widehat{y} \quad , \tag{4.14}$$

where $\widehat{z}$ is the Fourier spectrum of the signal z.

By using the cross-correlation measure we can finally construct the iterative WSOLA algorithm (Algorithm 3). Figure 4.5 visualizes one iteration of WSOLA.

4.3 Artifacts

The most typical artifacts occurring in audio signals that were produced by WSOLA are stuttering artifacts. Most often they are audible at transient regions in an audio signal. Transients are very short, noise-like parts in audio signals that typically originate from instrument onsets like for example the stroke of a guitar string or a drum hit. At

Algorithm 3: WSOLA

Data: DT signal x, window function w, overlap factor o, time-stretch function τ, window position tolerance Δ_{max}.

Result: DT y that is a time-scale modified version of x.

begin

`/* Compute` γ `*/`

$\gamma(1) = 1;$

for $i \leftarrow 2$ **to** $\lceil \frac{length(y)}{\eta_o^w} \rceil$ **do**

$\gamma(i) \longleftarrow \gamma(i-1) + \eta_o^w;$

`/* Compute` σ `*/`

for $i \leftarrow 1$ **to** *length*(γ) **do**

$\sigma(i) \longleftarrow \tau^{-1}(\gamma(i));$

$\Delta_1 \longleftarrow 0;$

for $k \leftarrow 2$ **to** *length*(σ) **do**

`/* Overlap and Add */`

$frame \longleftarrow x[b_w^{\sigma(k-1)+\Delta_{k-1}} : e_w^{\sigma(k)+\Delta_k}] \cdot w;$

$y[b_w^{\gamma(k-1)} : e_w^{\gamma(k-1)}] \longleftarrow y[b_w^{\gamma(k-1)} : e_w^{\gamma(k-1)}] + frame;$

`/* Find next offset */`

$\sigma'(k-1) \longleftarrow \sigma(k-1) + \Delta_{k-1} + \eta_o^w;$

$frameAdj \longleftarrow x[b_w^{\sigma'(k-1)} : e_w^{\sigma'(k-1)}];$

$frameNext \longleftarrow x[b_w^{\sigma(k)} - \Delta_{max} : e_w^{\sigma(k)} + \Delta_{max}]$

$\Delta_k \leftarrow$ getOffsetViaCrossCorrelation($frameAdj$,$frameNext$);

`/* Adjust possible amplitude modulations */`

$y \longleftarrow$ adjustAmplitude(y, w, o)

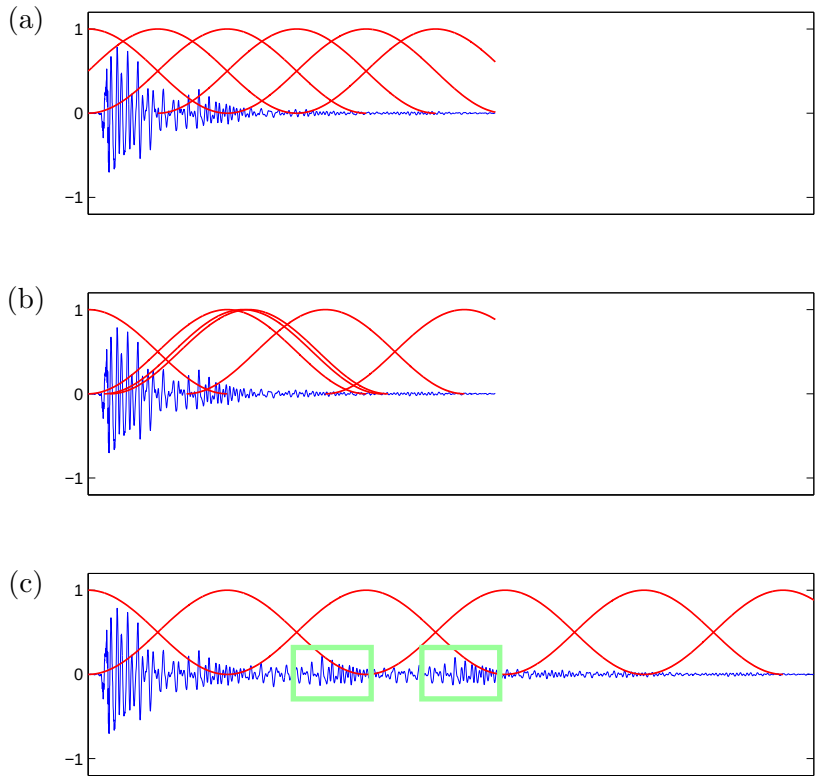

Figure 4.7. **(a)** The audio signal of a signal drum hit. The WSOLA input windows for a constant time-stretch of factor 2 are displayed. **(b)** The input windows after computing the offsets Δ_n for all windows. **(c)** The resulting audio signal. When listening to the audio signal one hears that the transient originating from the drum hit is repeated two times. The regions in the signal that are related to the repetitions are marked by green boxes.

such positions, a lot of energy is spread all over the frequency spectrum. Especially when stretching the signal at such a position with a large local time-stretch factor, the density of input window positions is high, and therefore this short moment of high energy is contained in several windowed audio segments. When WSOLA copies the windowed audio segments to the output, the short burst of energy is audible several times because of the temporal respacing of the windowed audio segments. Figure 4.7 shows an example of an occurrence of such a stuttering artifact.

Another interesting artifact occurs in case the wavelength of the fundamental frequency of the input signal x is at some point larger than the used window length. In this case a single windowed audio segment does not contain enough information to describe the local content of the signal since it is not possible to determine from this windowed audio segment the fundamental frequency that is contained in the signal. The task of finding a window offset that minimizes phase jumps in the fundamental frequency therefore degenerates to the task of just keeping the signal locally the same to the input signal. But we already saw that this approach does not yield the intended results and introduces pitch artifacts into the signal (see Equation 3.1 and Figure 3.2). An example of an instance in which this artifact occurs can be seen in Figure 4.8.

WSOLA has particular problems when time-scale modifying input signals which are highly

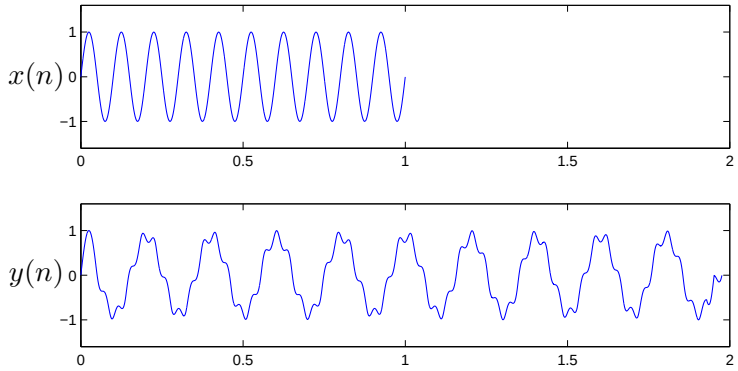

Figure 4.8. The input signal x is a sinusoidal signal with a frequency of 10 Hz, sampled at 22050 Hz. The signal is stretched using WSOLA by a factor of 2. The used window length is 2048 samples which is slightly less than the wavelength of the sinusoid which is 2205 samples. One can see that output signal y has a fundamental frequency of 5 Hz and is therefore pitch shifted.

polyphonic, like for example recordings of orchestral music. For such input signals the output often sounds *metallic* and *noisy*. This is due to the algorithm's core idea of choosing the input window positions such that phase jumps in the fundamental frequency of the signal are avoided. In case there is more than one fundamental frequency in the input signal, only one of them can be preserved properly. The phase continuity of the remaining fundamental frequencies is destroyed and many phase jumps are introduced into the output signal what results in the noisy sound.

A forth class of artifacts concerns the timbre of audio signals. The timbre of an instrument is not only dependent on the overtone spectrum of the sound but also on slight modulation effects introduced by the instrument. For example the typical sound of a violin has a *vibrato* (periodic modulation of the pitch of the signal) whereas the sound of a flute is influenced by a *tremolo* (periodic modulation of the amplitude of the signal). These effects, and especially the frequency of the modulation, are a natural component of the sound of those instruments. Manipulating the time-scale of a recording containing these instruments also alters the frequency of the vibrato or tremolo effect. As a result the sound of those instruments sounds unnatural (see Figure 4.9).

A related artifact can be observed if an instrument is recorded playing in a wide hall like for example in a church. The sound of the instrument is reflected on the walls of the hall and therefore perceived several times. This effect is called *reverb* and it is a natural part of our sound perception. In fact the human auditory system is able to estimate the size of a room based on the reverb of sound sources in the room. Stretching an audio recording with noticable reverb for example by a factor of two also doubles the delay times between the sound of the original source and its reverb. When listening to the stretched audio signal it therefore sounds as if the room in which the audio signal was recorded was larger than it actually is. These kind of artifacts are not only typical for WSOLA but for any TSM algorithm.

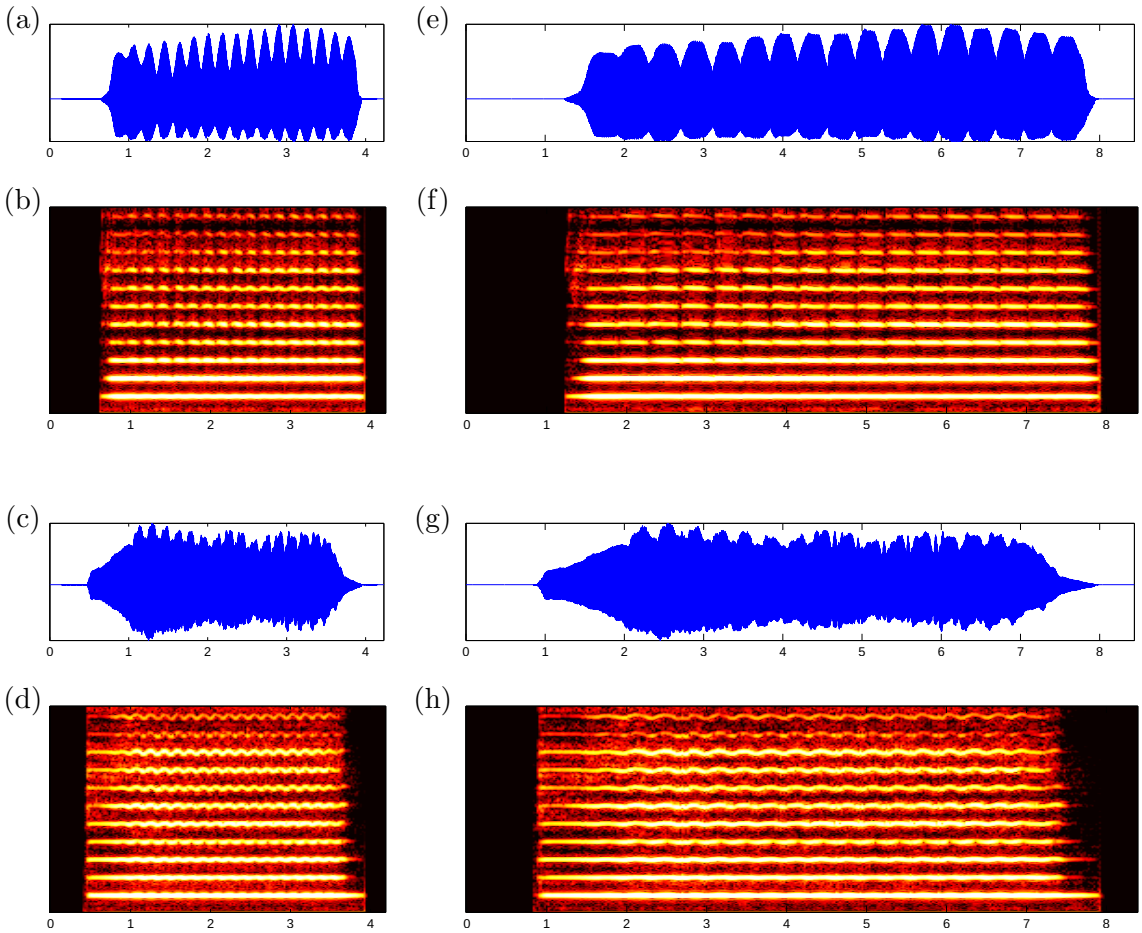

Figure 4.9. **(a)** The waveform of a flute playing a single note. **(b)** The spectrogram related to (a). The tremolo effect of the flute sound manifests itself as equidistant small vertical gaps in the energy distribution. **(c)** The waveform of a violin playing a single note. **(d)** The spectrogram related to (c). The vibrato effect of the violin sound can be observed as slight temporal pitch vibrations of a constant frequency. **(e)** The waveform from (a) stretched using WSOLA by a factor of two. **(f)** The spectrum related to (e). Note that the fundamental frequency of the signal as well as its overtone spectrum stayed untouched whereas the gaps in the spectrum that originated from the tremolo were respaced. This leads to a tremolo that is half as fast as in the original signal. **(g)** The waveform from (c) stretched using WSOLA by a factor of two. **(h)** The spectrogram related to (g). Again note that the frequency content was not altered during the time-scale modification but the vibrato is now on half the original tempo.

Chapter 5

Phase Vocoder

The most prominent frequency-domain TSM algorithm is the Phase Vocoder [14, 1, 9, 10, 42]. It was initially designed to encode the sound of the human voice (therefore the abbreviation *vocoder* for *voice encoder*) such that in a communication scenario as little data as possible needs to be transmitted. Nevertheless, it showed that it was also possible to alter the time-scale of the encoded signal by manipulating the encoding. The most important tool of the Phase Vocoder is the Short Time Fourier Transform which we discuss in Section 5.1. Afterwards we describe the high level pipeline of the algorithm in Section 5.2 whereas the core technique of the Phase Vocoder, the so called *phase propagation* is discussed in Section 5.3. Finally in Section 5.4 we develop some modifications to the algorithm, which are based on the Phase Vocoder implementation of [12], that make the implementation easier and close with a brief analysis of occurring artifacts in Section 5.5.

5.1 Short Time Fourier Transform

The Short Time Fourier Transform, or short STFT, is an algorithm to analyze a signal in terms of frequency composition while at the same time keeping some temporal information, and the most important tool of the Phase Vocoder. We already introduced it briefly in Equation 4.4 but we will restate it here for the sake of convenience. The STFT is defined as

$$X(t,k) = \sum_{n \in \mathbb{Z}} x(t+n) \cdot w(n) \cdot e^{-\frac{2 \cdot \pi \cdot i \cdot k \cdot n}{N}} \quad , \tag{5.1}$$

where t is the *analysis instance* given in samples, w is the analysis window function, N is the size of the discrete Fourier Transform and and k is the index of the frequency bin with center frequency $\frac{2 \cdot \pi \cdot k}{N}$. In the context of the Phase Vocoder a frequency bin is also called a *vocoder channel*. Further we define that

$$X_t = X(t, \cdot) \tag{5.2}$$

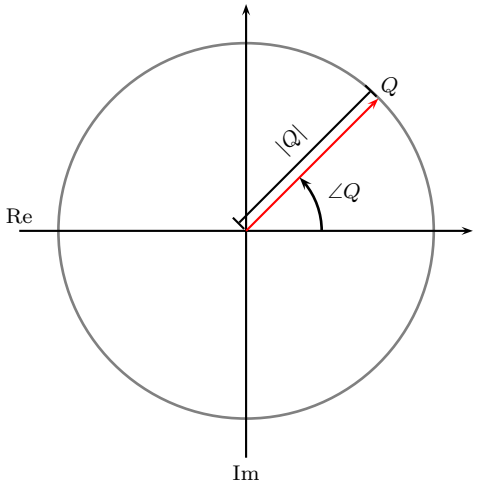

Figure 5.1. The complex value $Q = X(t,k)$ can be seen as a point given in rectangular coordinates $(\mathrm{Re}(Q), \mathrm{Im}(Q))$. To get the phase and the magnitude of the partial in vocoder channel with index k at analysis instance t we change the representation to polar coordinates. We compute the phase by $\angle Q = \arctan\left(\frac{\mathrm{Im}(Q)}{\mathrm{Re}(Q)}\right)$ and the magnitude $|Q| = \sqrt{(\mathrm{Re}(Q))^2 + (\mathrm{Im}(Q))^2}$. It always holds that the phase $\angle Q \in [-\pi, \pi)$.

is the Fourier spectrum at analysis instance t and

$$X_\nu = (X_{\nu(1)}, X_{\nu(2)}, ..., X_{\nu(M)}) \tag{5.3}$$

for some vector of analysis instances ν of length M. We write $X_\nu(n)$ to denote the n^{th} spectrum in the sequence X_ν and $X_\nu(n,k)$ for its value in the vocoder channel with index k.

Note that $X(t,k)$ for some analysis instance t and and vocoder channel index k is a complex value. The magnitude $|X(t,k)|$ quantifies how much the partial in the vocoder channel with center frequency $\frac{2\cdot\pi\cdot k}{N}$ contributes to the signal x around the analysis instance t. It is computed as

$$|X(t,k)| = \sqrt{(\mathrm{Re}(X(t,k)))^2 + (\mathrm{Im}(X(t,k)))^2} \quad . \tag{5.4}$$

Furthermore the complex value $X(t,k)$ also encodes the phase of that partial. This phase is denoted by $\angle X(t,k)$ and can be computed as

$$\angle X(t,k) = \arctan\left(\frac{\mathrm{Im}(X(t,k))}{\mathrm{Re}(X(t,k))}\right) \quad . \tag{5.5}$$

See Figure 5.1 for an example. The operators $|\cdot|$ and $\angle\cdot$ may also be applied to matrices where the application is defined element-wise.

We now come to the inverse of the STFT. Typically the STFT is applied to an audio signal x using a vector of analysis instances α. This yields the sequence of Fourier spectra X_α.

But in the context of the Phase Vocoder it is very likely that we do not want to apply the inverse Short Time Fourier transform (iSTFT) to X_α directly but to some modification of it. We therefore generalize the iSTFT. Let $Z \in \mathbb{C}^{M \times N}$ be some sequence of Fourier spectra of length M. Note that Z does not need to be the STFT of any audio signal. To invert the STFT we apply the inverse Fourier Transform to each Fourier spectrum in Z and concatenate the resulting audio fragments. We can compute the j^{th} audio fragment of the output signal y by

$$y_j(n) = \mathrm{Re}\left(\frac{1}{N}\sum_{k=0}^{N-1} Z(j,k) \cdot e^{\frac{2 \cdot \pi \cdot i \cdot k \cdot n}{N}}\right) \quad . \tag{5.6}$$

From these partial signals we can construct the output by a simple overlap and add. To avoid modulation artifacts, a synthesis window w is applied to the signal fragments before adding them up. Let β be a vector of *synthesis instances* given in samples which is of length M. The vector β defines where the audio fragments that are computed by Equation 5.6 are placed in the output signal.

$$y(n) = \sum_{m=1}^{M} w(n - \beta(m)) \cdot y_m(n - \beta(m) + \left\lceil \frac{w_\ell}{2} \right\rceil + 1) \quad . \tag{5.7}$$

In case that $Z = X(\alpha, k)$ is the STFT of a signal x for synthesis instances α and it holds that $\alpha = \beta$, the signal x and the signal y are identical up to minor artifacts introduced by the used synthesis window w.

5.2 Phase Vocoder Pipeline

Given to the Phase Vocoder are an input signal x together with a time-stretch function τ. The main idea of the Phase Vocoder is to construct a sequence of appropriate Fourier spectra from x such that applying the iSTFT to this sequence yields the intended time-scale modification of x. Although it may not appear obvious at first sight, the basic mechanics of the Phase Vocoder are similar to those of WSOLA: it strives to construct a sequence of audio fragments from the input signal such that overlap-adding those audio fragments yields an output signal that is the time-scale modified input signal. The difference is that the Phase Vocoder preserves phase continuity not by small modifications of the fragment positions but by manipulating the Fourier spectra of the fragments such that the phases of consecutive signal fragments match. Nevertheless the basic mechanics of WSOLA and the Phase Vocoder have a lot in common. Therefore also the internal representations of the time-stretch function τ are similar in both algorithms. While WSOLA computes an output window position vector γ and an input position vector σ from τ, the Phase Vocoder produces a vector of synthesis instances β and a vector of analysis instances α. For the

moment the vectors α and β are computed exactly like σ and γ respectively, namely by

$$\begin{array}{ll} (i) & length(\beta) = \lceil \frac{length(y)}{\eta_o^w} \rceil \\ (ii) & \beta(1) = 1 \\ (iii) & \beta(n) = \beta(n-1) + \eta_o^w \text{ for } n \in [2 : length(\beta)] \end{array} \tag{5.8}$$

and

$$\alpha(n) = \tau^{-1}(\beta(n)) \tag{5.9}$$

for all $n \in [1 : length(\beta)]$. Note that w describes the analysis window of the STFT in this case and the overlap factor o is some fixed constant. Nevertheless we will discuss some changes to the Phase Vocoder which will simplify its implementation but will also affect the definitions of α and β in Section 5.4.

The first step of the algorithm is to compute the STFT of x at the analysis instances α. This yields the sequence of frequency spectra X_α. Unfortunately if we now compute the output signal y by applying the iSTFT to X_α using the synthesis instances β we run into the same problems as with the basic OLA technique: by not taking care of the phases of the partials we get phase jump artifacts in the output. Therefore we have to adjust the phases of all partials in the Fourier spectra X_α before applying the iSTFT. This step involves a technique called *phase propagation* which is discussed in detail in Section 5.3. It yields the phase adjusted sequence of Fourier spectra which we denote by $\widetilde{X}$. Finally, applying the iSTFT to $\widetilde{X}$ using the synthesis instances β yields the time-scale modified output signal y. The several stages of the Phase Vocoder are visualized in Figure 5.2.

The Phase Vocoder realizes time-scale modification of audio signals by maintaining local similarity of the magnitude spectra of the input and output spectra and at the same time keeping the phase of all partials of the output signal continuous. It is therefore, like WSOLA, an approach to fulfill the instantiation of $\approx$ given in Equation 4.8.

5.3 Phase Propagation

Having a sequence of Fourier spectra X_α we now need to think about how to change the phases of all the partials in all vocoder channels such that no phase jumps occur when copying the audio signal fragments, that result from applying the inverse Fourier Transform to the spectra, to the output signal at synthesis instances β. The core idea is the following: Given a discrete sinusoidal wave with angular frequency ω and phase φ at some sample point n. We now want to know which phase this sinusoidal wave will have at point m where $m > n$. But knowing the frequency ω and the phase φ makes this simple. The value $(m-n) \cdot \omega$ gives us the phase increment of the sinusoidal wave from point n to point m. Therefore the phase of the sinusoidal wave at m will be $\varphi + (m-n) \cdot \omega$ (we need to shift this value to the range of $[-\pi, \pi)$ such that it is a valid phase). This is exactly what we want to know in the Phase Vocoder: For our first spectrum we do not need to adapt any phases and we can therefore set $\widetilde{X}(1) = X_\alpha(1)$. For $\widetilde{X}(2)$ we already know

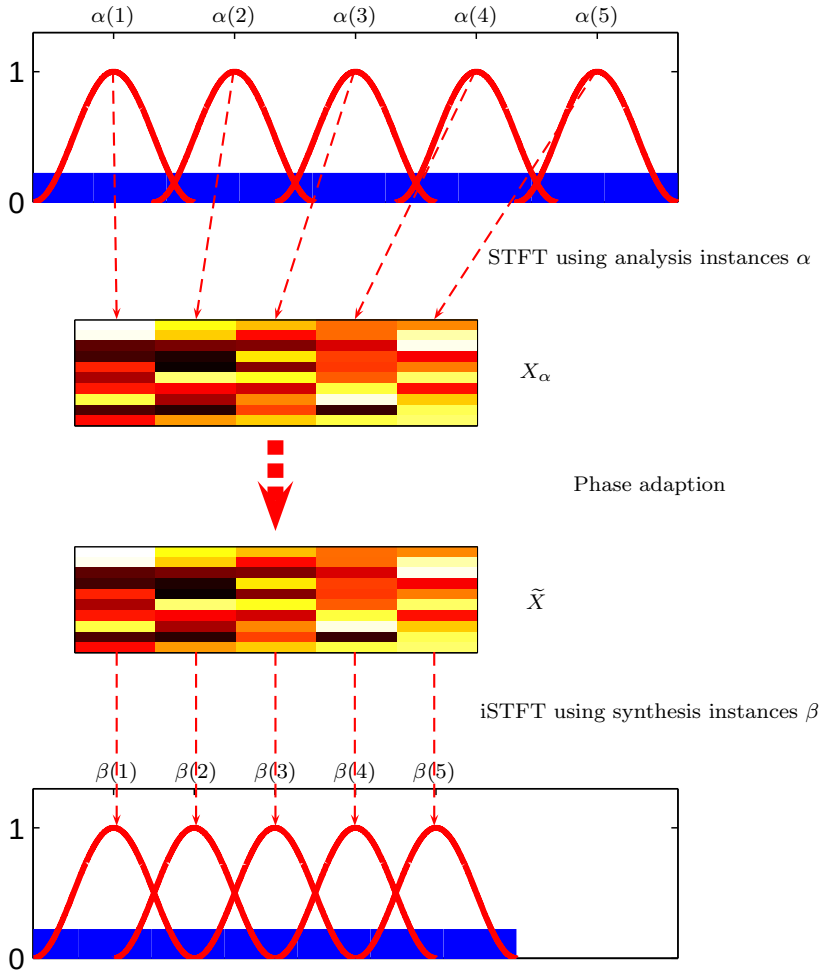

Figure 5.2. Schematic mechanics of the Phase Vocoder. The blue bars symbolize the audio signals. The spectrograms X_α and $\widetilde{X}$ are schematic. The input signal is windowed at the analysis instances specified by α. All windowed audio segments are then Fourier transformed. After the phase adjustment all spectra are transformed back to the time-domain using the inverse Fourier Transform, windowed and copied to the output at synthesis instances β. The example shows how the Phase Vocoder produces a version of the input signal that is speeded up by a constant factor.

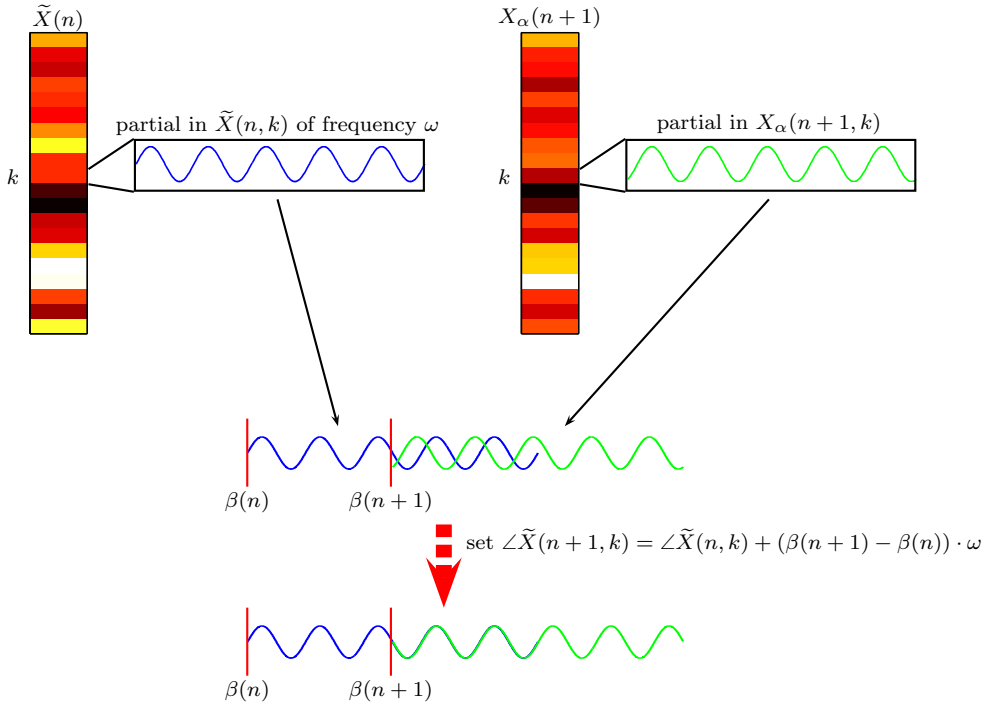

Figure 5.3. Visualization of the phase propagation. For this example we investigate a single partial, but the phase propagation works in the same way for all partials in all vocoder channels. Assume that we already have already computed $\widetilde{X}(n)$ correctly. The goal is to set the phase of the green sinusoid to the phase of the blue sinusoid at $\beta(n+1)$. Since we know that the phase of the blue sinusoid is initially $\angle\widetilde{X}(n,k)$ we can compute its phase at $\beta(n+1)$ as $\angle\widetilde{X}(n,k)+(\beta(n+1)-\beta(n))\cdot\omega$ and therefore can set the phase of $\widetilde{X}(n+1,k)$ correctly. The phase is propagated from $\widetilde{X}(n)$ to $\widetilde{X}(n+1)$.

the magnitude spectrum $|\widetilde{X}(2)| = |X_\alpha(2)|$. To compute $\angle\widetilde{X}(2)$ we can proceed exactly as mentioned above for every single vocoder channel k. In this case the points n and m are $\beta(1)$ and $\beta(2)$. This technique is called *phase propagation* since the phases of a Fourier spectrum in a sequence of Fourier spectra are propagated to the next spectrum in the sequence. The process is visualized in Figure 5.3.

The problem is that we do not know the frequencies of all the partials in the vocoder channels exactly. Let $\Omega_k = \frac{2\cdot\pi\cdot k}{N}$ be the center frequency of the k^{th} vocoder channel. A partial lying in a phase vocoder channel with center frequency Ω_k not necessarily has the frequency Ω_k by itself. It just has a frequency that is closer to Ω_k as to any other center frequency of the remaining vocoder channels. We therefore have to estimate the exact frequencies of all the partials first. To this end we compute the so called *heterodyned phase increment* of vocoder channel k at analysis instance $\alpha(n)$ which is denoted by $\Delta\phi_k^{\alpha(n)}$.

$$\Delta\phi_k^{\alpha(n)} = \angle X_\alpha(n,k) - \angle X_\alpha(n-1,k) - (\alpha(n) - \alpha(n-1)) \cdot \Omega_k \quad . \tag{5.10}$$

The term $\angle X_\alpha(n,k) - \angle X_\alpha(n-1,k)$ describes the phase increment from analysis instance $\alpha(n-1)$ to $\alpha(n)$ that is actually present in the Fourier spectra. The term $(\alpha(n) - \alpha(n-1)) \cdot \Omega_k$ on the other hand describes the expected phase increment in case the partial in vocoder channel k had exactly frequency Ω_k. The heterodyned phase increment therefore describes the small deviation of the computed phase increment from the phase increment that a partial with frequency exactly Ω_k would have had. Note that we have to shift the heterodyned phase increment $\Delta\phi_k^{\alpha(n)}$ to the range $[-\pi, \pi)$ such that it is a valid phase. For the sake of simplicity we will denote the shifted version by $\Delta\phi_k$.

From $\Delta\phi_k$ we now can compute the actual frequency of the partial that lies in vocoder channel k at analysis instance $\alpha(n)$. This frequency is called the *instantaneous frequency* of the partial at position $\alpha(n)$ and is denoted by $\hat{\omega}_k(\alpha(n))$. Note that dividing a phase φ by some distance d yields the frequency of the sinusoidal wave that has phase φ at d.

$$\hat{\omega}_k(\alpha(n)) = \Omega_k + \frac{\Delta\phi_k}{\alpha(n) - \alpha(n-1)} \quad . \tag{5.11}$$

Finally having estimated the exact frequencies we can apply the phase propagation by

$$\angle\widetilde{X}(n,k) = \angle\widetilde{X}(n-1,k) + (\beta(n) - \beta(n-1)) \cdot \hat{\omega}_k(\alpha(n)) \quad . \tag{5.12}$$

5.4 Modifications for a simple implementation

In implementations of the Phase Vocoder it is often convenient to keep the rather complex phase propagation as simple as possible. To this end we can modify the pipeline described in 5.2 a little bit. Instead of achieving the time-scale modification by spacing the analysis instances of the STFT differently from the synthesis instances we can also construct a completely new sequence of Fourier spectra for the output signal from the STFT of the input signal. The core idea of the new approach, and the reason why the phase propagation

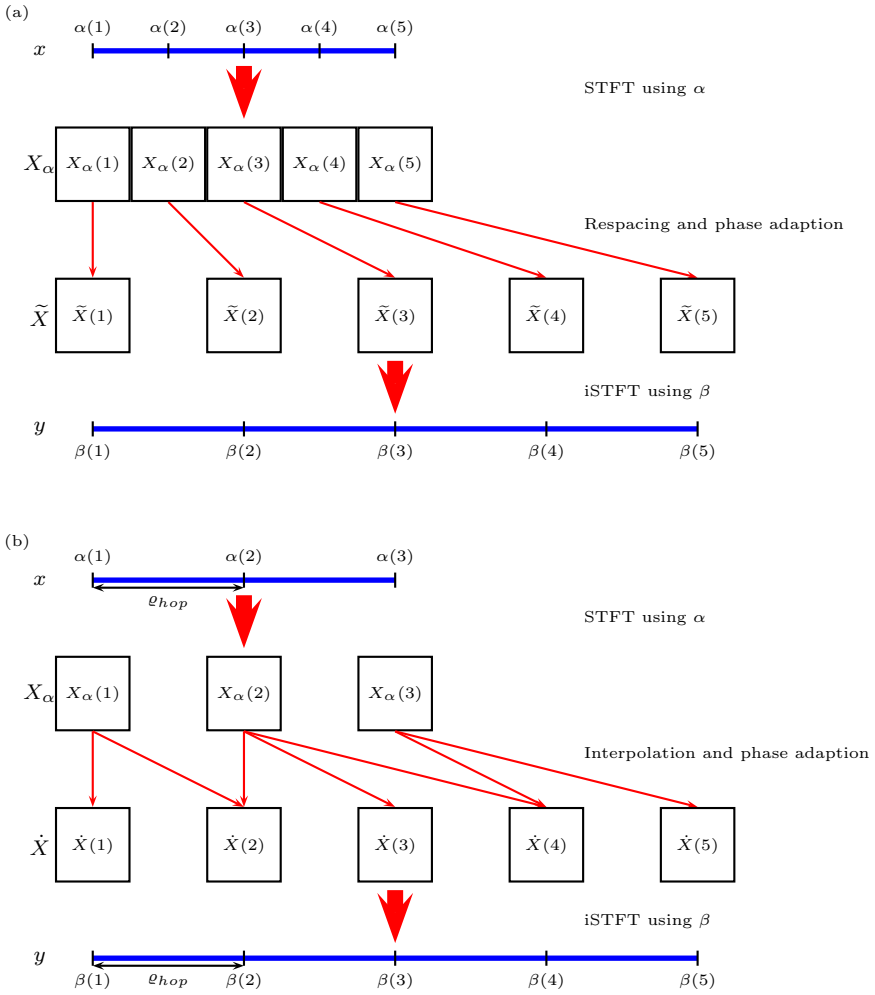

Figure 5.4. **(a)** Schematic overview of the standard Phase Vocoder. Note that the vectors α and β are of the same length. **(b)** Schematic overview of the modified Phase Vocoder for a simpler implementation. In opposite to the standard Phase Vocoder the Fourier Spectra for the output signal y are not directly computed from the input signal x but interpolated from some spectra that were computed at equidistant analysis instances α. Note that therefore the length of the vector of analysis instances α is unequal to the length of the vector of synthesis instances β. Note further that all analysis instance and synthesis instances are spaced apart by the same distance ϱ_{hop}

becomes simple in the end, is to have equidistant analysis and synthesis instances, all spaced apart by a constant value. This value is called the *hop factor* and is denoted by ϱ_{hop}. Note that therefore the vectors α and β do not need to be of the same length any more. In fact whenever the duration of a time-scale modified signal differs from the duration of the initial signal their length will also differ. Figure 5.4 visualizes the differences between the two Phase Vocoder approaches.

For a given input signal x and a time-stretch function τ we can define

$$\begin{array}{ll} (i) & length(\alpha) = \lceil \frac{length(x)}{\varrho_{hop}} \rceil \\ (ii) & \alpha(1) = 1 \\ (iii) & \alpha(n) = \alpha(n-1) + \varrho_{hop} \text{ for } n \in [2 : length(\alpha)] \end{array} \tag{5.13}$$

$$\begin{array}{ll} (i) & length(\beta) = \lceil \frac{length(y)}{\varrho_{hop}} \rceil \\ (ii) & \beta(1) = 1 \\ (iii) & \beta(n) = \beta(n-1) + \varrho_{hop} \text{ for } n \in [2 : length(\beta)] \end{array} \tag{5.14}$$

Applying the STFT to the input signal x at analysis instances α yields X_α. Our goal is now to construct from X_α a sequence of Fourier spectra $\dot{X}$ of length $length(\beta)$ such that when applying the inverse Fourier Transform to $\dot{X}$ using synthesis instances β we get the desired output signal. We start by first constructing a sequence of Fourier spectra $\ddot{X}$ that has the same magnitude spectrum as $\dot{X}$ such that it holds that $|\dot{X}| = |\ddot{X}|$. Let $f_\alpha^x : [1 : length(x)] \to \mathbb{C}^N$ be a function defined as follows.

$$f_\alpha^x(t) = \tfrac{t-\alpha(i)}{\alpha(i+1)-\alpha(i)} \cdot X_\alpha(i) + \tfrac{\alpha(i+1)-t}{\alpha(i+1)-\alpha(i)} \cdot X_\alpha(i+1) \quad \text{for } \alpha(i) \leq t \leq \alpha(i+1) \quad . \tag{5.15}$$

The function f_α^x assigns to every sample position in x a Fourier spectrum that was computed by linear interpolation between the Fourier spectra of x at analysis instances α. We therefore can compute

$$\ddot{X}(n) = f_\alpha^x(\tau^{-1}(\beta(n))) \quad . \tag{5.16}$$

The internal encoding of the time-stretch function τ therefore does not lie in the vectors of analysis and synthesis instances anymore, but in the sequence of Fourier Spectra itself.

Having computed $\ddot{X}$ and knowing that $|\dot{X}| = |\ddot{X}|$ we now have to take care of the phases $\angle\dot{X}$. This is again done using the phase propagation technique. Recall that to compute the phase propagation we have to know the exact frequencies of all the partials lying in the vocoder channels. To compute these exact frequencies we can use the phases of two successive Fourier spectra. Unfortunately we can not assume the phases of $\ddot{X}$ to be meaningful since $\ddot{X}$ was computed by linear interpolation. The only source of reliable phase information is the sequence of Fourier spectra X_α. We therefore assume that all Fourier spectra in $\ddot{X}$ that were computed by linear interpolation between $X_\alpha(n-1)$ and $X_\alpha(n)$ have those exact frequencies that can be computed using $X_\alpha(n-1)$ and $X_\alpha(n)$. To

be able to compute those exact frequencies we need to be able to access these two spectra given the index of a synthesis instance in β. Note that the term

$$g_m = \alpha\left(\left\lceil \frac{\tau^{-1}(\beta(m))}{\varrho_{hop}} \right\rceil\right) \tag{5.17}$$

yields the index i in α such that $\alpha(i)$ is the biggest entry in α that is smaller than $\tau^{-1}(\beta(m))$ since the difference between two analysis or synthesis instances is by construction always constantly ϱ_{hop}. Furthermore $g_m + 1$ is the index of the smallest entry in α that is larger than $\tau^{-1}(\beta(m))$. Therefore we can define

$$X_1(m) = X_\alpha(g_m) \tag{5.18}$$

and

$$X_2(m) = X_\alpha(g_m + 1) \quad . \tag{5.19}$$

$X_1(m)$ and $X_2(m)$ are exactly the two spectra that were used to interpolate $\ddot{X}(m)$.

Finally we can compute the phase propagation as

$$\begin{aligned}
\angle\dot{X}(n,k) &= \angle\dot{X}(n-1,k) + \varrho_{hop}\cdot\hat{\omega}_k(\alpha(n)) && (5.12)\\
&= \angle\dot{X}(n-1,k) + \varrho_{hop}\cdot(\Omega_k + \tfrac{\Delta\phi_k}{\varrho_{hop}}) && (5.11)\\
&= \angle\dot{X}(n-1,k) + \varrho_{hop}\cdot\Omega_k + \Delta\phi_k\\
&= \angle\dot{X}(n-1,k) + \varrho_{hop}\cdot\Omega_k\\
&\quad +(\angle X_2(n,k) - \angle X_1(n,k) - \varrho_{hop}\cdot\Omega_k) && (5.10)\\
&= \angle\dot{X}(n-1,k) + (\angle X_1(n,k) - \angle X_2(n,k))
\end{aligned}$$

In other words, whenever we want to calculate the phase $\angle\dot{X}(n)$ for a magnitude spectrum $|\dot{X}(n)|$ that was constructing by linear interpolation between two Fourier spectra $X_1(n)$ and $X_2(n)$ we simply have to add the phase differences $\angle X_2(n) - \angle X_1(n)$ to the known phases $\angle\dot{X}(n-1)$ (where $\angle\dot{X}(1) = X_\alpha(1),$). Note that in the computations above we argued about a single vocoder channel. But since the computations are the same for every vocoder channel we can do the computations on the whole spectra directly.

We get the time-scale modified output signal y by applying the iSTFT to $\dot{X}$ using the analysis instances β. The modified pipeline is visualized in Figure 5.5.

5.5 Artifacts

When listening to audio signals produced by the Phase Vocoder they often sound like if the content of the original signal was played in some distance. Percussive elements in the signal sound hollow and also non percussive parts sound phasy. These artifacts occur not only when applying large time-stretch factors but immediately when time-scale modifying the signal only slightly. The cause of these artifacts lies in the destruction of the *vertical phase coherence* of the signal. By design, the Phase Vocoder ensures *horizontal phase*

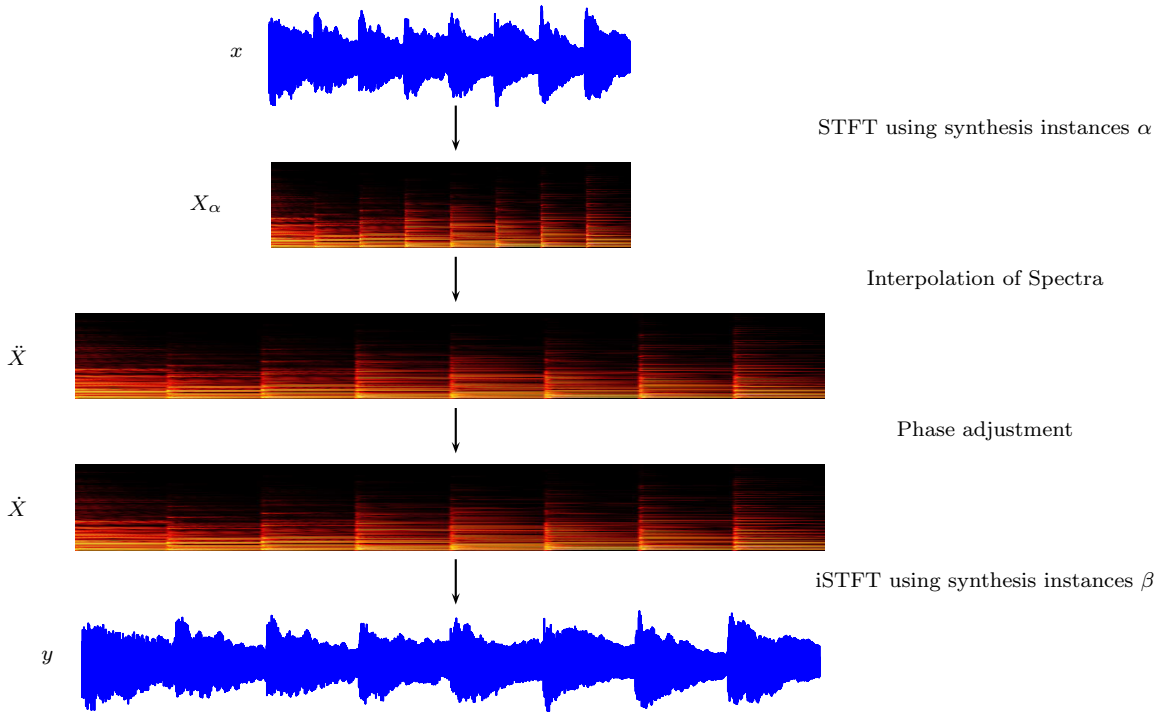

Figure 5.5. The pipeline of the modified Phase Vocoder. In this example the Phase Vocoder is applied with a time-stretch factor of 2. Note that the length of $\ddot{X}$ is twice the length of X_α.

coherence, meaning that over the duration of the signal the phases in a vocoder channel are adapted such that there occur no phase jumps in this particular vocoder channel. But for the overall sound of the signal also the relations of the phases in the several vocoder channels at a given point in time are important. These relations are referred to as vertical phase coherence and they are not maintained by the Phase Vocoder. Figures 5.6 and Figure 5.7 show examples how the loss of vertical phase coherence affects an audio signal.

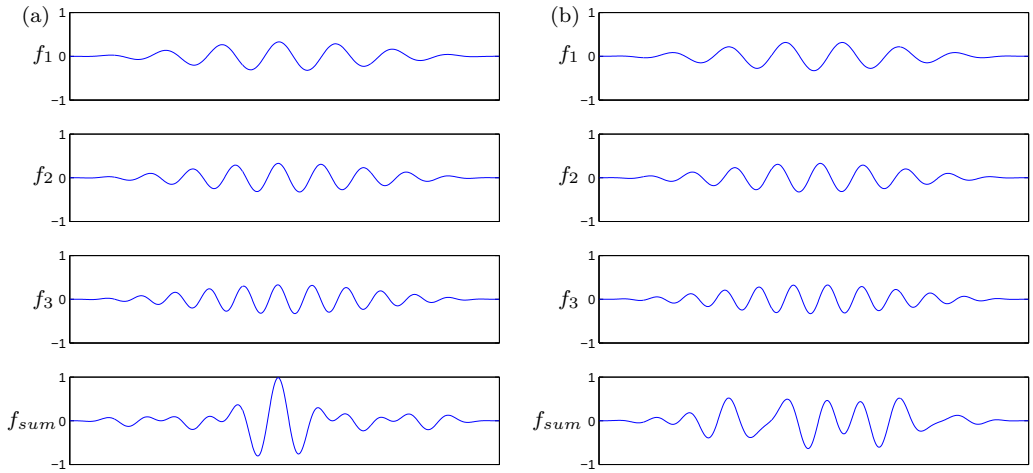

Figure 5.6. **(a)** The three partials f_1, f_2 and f_3 form the signal f_{sum}. The signal f_{sum} has a clear peak structure. **(b)** The frequencies of the partials f_1, f_2 and f_3 are the same as in (a), but the phases are slightly modified. The signal f_{sum} lost its peak structure because of the destroyed vertical phase coherence.

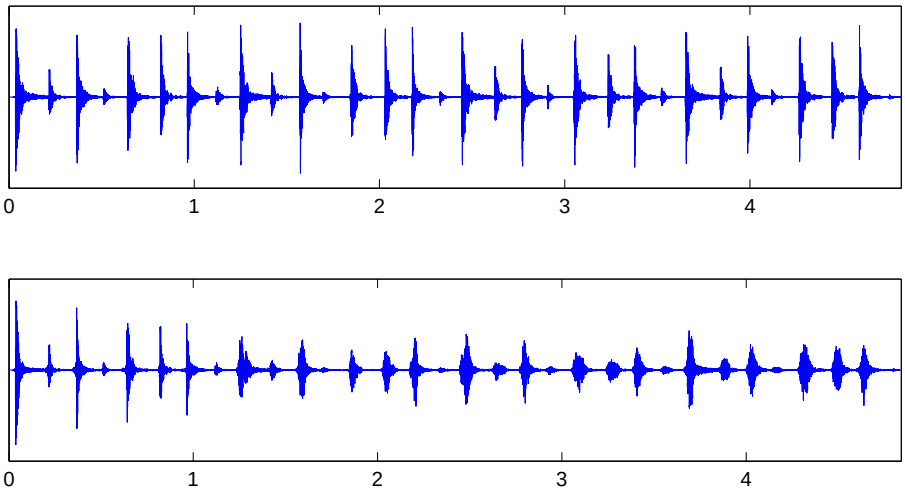

Figure 5.7. The audio signal of drum beat and its time-scale modified version produced with the Phase Vocoder. The constant time-stretch factor used in this example is 1.01. One can observe that the amplitude of the peaks decreased in the time-scale modified version. Note that the amplitude of the first beats were only slightly decreased. This is due to the fact that we initialized $\widetilde{X}(1) = X_\alpha(1)$ and the deviation from the initial vertical phase coherence is therefore smaller at the beginning of the signal.

Chapter 6

Transient Preserving WSOLA

As noted in Section 4.3 stuttering artifacts at transients are a significant cause for audio signal quality degradation in WSOLA. A method to prevent this kind of degradation is the so called *transient preservation*. The core idea is to suspend the time-scale modification at transient positions in the audio signal. To compensate for these suspensions the remaining parts of the signal are stretched with a slightly different time-stretch factor. This can be done by manipulations of the time-stretch function τ. In this chapter we first introduce a novel, simple and effective way of representing time-stretch functions in Section 6.1. Afterwards we will discuss the transient preservation process in detail: We start with some basic transient detection in Section 6.2. Knowing the positions of the transients in an audio signal we can proceed with the transient preservation by manipulating the time-stretch function. The process is explained in detail in Section 6.3. Afterwards we will investigate the limits of transient preservation in Section 6.4.

6.1 Anchor Points

In practical applications it is necessary to have a simple and practical representation of the time-stretch function τ as an input for the WSOLA algorithm. We therefore represent it by a set of so called *anchor points*. An anchor point is a pair of two points in time where the first entry specifies a point in time in the input audio signal and the second entry specifies where this point in time should be mapped to in the output audio signal. Between two anchor points the audio signal is modified in a linear fashion. Using anchor points is a simple way of expressing piecewise linear time-stretch functions which are sufficient for most applications. For example in the context of the Connector where we want to rearrange the beat positions of an audio signal, we have exactly one anchor point per beat. The first entry of the anchor point is set to the time of the beat in the input audio signal and the second entry to the time where the beat should be placed. An example can be seen in Figure 6.1.

In fact, also constant time-stretch factors are realized using anchor points. In case our WSOLA implementation receives a global time-stretch factor t as an input along with an audio signal x, the implementation just introduces two anchor points: one to synchronize

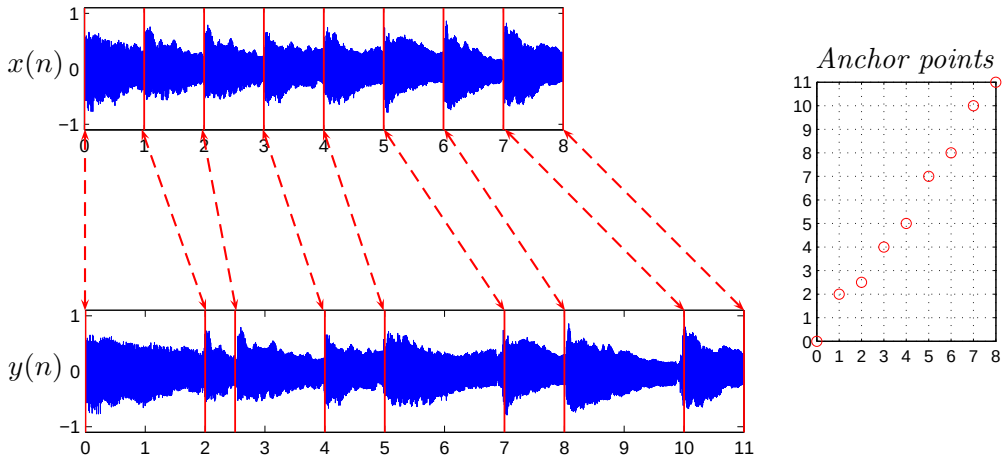

Figure 6.1. An audio signal x and its time-scale modified version y (time is given in seconds). The non-linear time-scale modification was specified using a set of anchor points. For each beat in x (indicated by red bars) there exists an anchor point that specifies the point in time where this beat should be mapped to in the output. Note that the red bars in y mark exactly the beat positions again.

the beginnings of input and output (this is the anchor point $(1, 1)$ that maps sample position 1 in the input to sample position 1 in the output) and one to synchronize the endings (which is $(length(x), t \cdot length(x))$ since the output should have t times the length of the input).

To compute the input position vector σ we interpolate τ from the set of anchor points by discrete linear interpolation (recall from Chapter 4 that σ is computed by $\sigma(n) = \tau^{-1}(\gamma(n))$. This was defined in Equation 4.2). Note that in principle the anchor points form exactly a discretized time-stretch function according to Definition 3.4. To compute σ we then can use the inverted interpolated time-stretch function accoring to Definition 3.6.

6.2 Transient Detection

Not every transient in an audio signal causes a stuttering artifact when time-scale manipulating the signal with WSOLA. These artifacts are for example rarely audible when time-scale modifying a recording of a violin. On the other hand they are almost always present when stretching the recording of a drum. To find the transients that produce the undesired stuttering artifacts in WSOLA it is therefore necessary to first describe the class of artifact causing transients.

According to our experience the most perceivable stuttering artifacts occur at points in the signal where some source produces a short, noise like burst of energy. These bursts are then audible several times in the time-scale modified signal what causes the stuttering sound. Sound sources which frequently produce such artifacts are for example drum or

cymbal hits but also strong onsets of instruments that have a percussive component like a piano. In general we can say that we are looking for strong *note onsets* of both percussive and non-percussive instruments. A note onset is the point in an audio signal where the perception of the note starts and it is commonly marks the beginning of a transient. See Figure 6.2 for an example. Unfortunately the task of detecting onsets in an audio signal is by far not trivial. A lot of work has been done in this field (for example [20, 2, 5, 4, 27, 7]), but the task of finding all note onsets in an arbitrary audio signal is not solved yet. The reason for this is that a perfect onset detection algorithm should be able to detect even the softest onsets like for example those of softly played violins. This is extremely difficult. Fortunately we do not need a perfect onset detection algorithm. We are only interested in onsets that are well perceivable as such since stuttering artifacts are not likely to occur at soft note onsets. The task of finding this kind of note onsets is much simpler. We are looking for massive bursts of energy in the frequency content of the signal. These can be found by looking for peaks in the sum of the magnitudes of all frequency bands. We also feel that the high frequency bands contribute more to noticeable transients than the low frequency bands and therefore weight the high frequency content more than the low frequencies. To this end we combine two standard techniques and apply them to the audio signal. Let x be the input audio signal. We first introduce the *high frequency content* of x [4]

$$HFC_x(n) = \sum_{k=2}^{N/2} k \cdot |X_\alpha(n)|^2 \quad , \tag{6.1}$$

where α is a vector of equidistant analysis instances in x of length M, $n \in [1 : M]$ and X_α is the Short Time Fourier Transform of x according to the definitions introduced in Chapter 5. The high frequency content is a function that describes the instantaneous energy of the signal at a given point in time while the energy in high frequency bands is weighted more than the energy in low frequency bands.

A standard technique to find bursts of energy in an audio signal is the so called *spectral difference* [2]. It is defined by

$$SD_x(n) = \sum_{k=1}^{N/2} (|X_\alpha(n)| - |X_\alpha(n-1)|)^2 \quad . \tag{6.2}$$

In principle $SD_x(n)$ computes a simple derivative of every single frequency band and sums up the values to get a one dimensional function. A burst in energy in the signal means a massive increase of the values in many frequency bands. Therefore the derivative is high in many frequency bands at the same time and $SD_x(n)$ yields high values at this point.

Finally we combine the high frequency content with the spectral difference by weighting the frequency bands in the spectral difference similar to the high frequency content. We applied a logarithmic compression function to the Fourier spectra to account for the

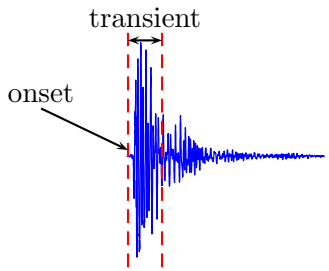

Figure 6.2. Note onset and transient according to [2].

logarithmic perception of loudness of the human ear.

$$D_x(n) = \sum_{k=1}^{N/2} k \cdot (|\log(1 + C \cdot X_\alpha(n))| - |\log(1 + C \cdot X_\alpha(n-1)|) \tag{6.3}$$

with

$$D_x(1) = \sum_{k=1}^{N/2} k \cdot |\log(1 + C \cdot X_\alpha(1))| \quad . \tag{6.4}$$

We set $C = 100$ in our experiments. In the function D_x we already have peaks at all the interesting transient positions in the input signal, but we also have a lot of noise and negative values that we do not need. The noise comes from smaller energy variations that do not originate from transients. The negative values indicate abrupt drops of the energy. Since we want to focus on note onsets, sudden drops of energy are not of interest to us. We therefore introduce a threshold function λ. Let $G^s_{D_x}$ be a version of D_x smoothed with a Hann-window of a length that is equivalent to s seconds in x. Furthermore, let v be the value of the global maximum of D_x. We define λ by

$$\lambda(n) = \kappa_{local} \cdot G^s_{D_x}(n) + \kappa_{global} \cdot v \quad , \tag{6.5}$$

where $\kappa_{local}, \kappa_{global} \in \mathbb{R}$ are variables to control the influence of the two parts. The threshold function λ is therefore a combination of a local average $G^s_{D_x}$ and a global absolute threshold $\kappa \cdot v$. We then compute

$$T_x(n) = \max(0, D_x(n) - \lambda(n)) \quad . \tag{6.6}$$

We call the function T_x a *novelty curve* of the signal x. Figure 6.3 shows the audio signal of a musical scale played on a piano together with its novelty curve.

It is important to notice that our transient detection method is not meant to be a state of the art onset detector. Modern onset detectors tend to be very sensitive to the input audio signal and often yield false positives. Furthermore, they are often capable of detecting

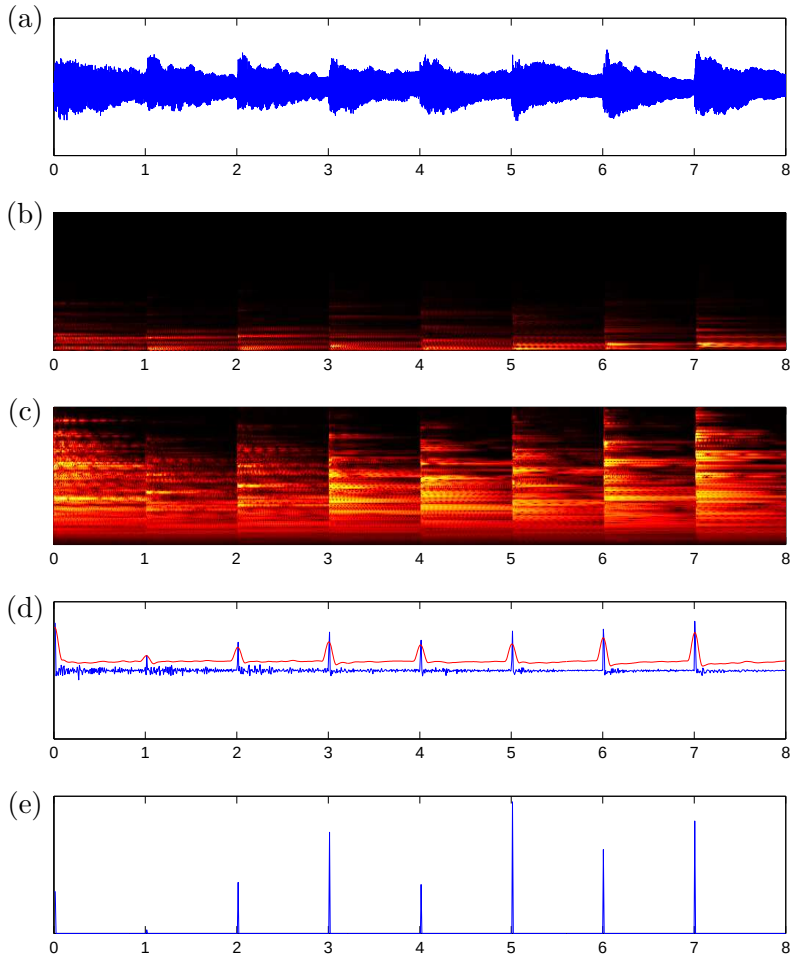

Figure 6.3. **(a)** The waveform of an audio signal x of a musical scale played on a piano. **(b)** The Short Time Fourier Transform X_α for a vector of equidistant analysis instances α. **(c)** The Short Time Fourier Transform from (b) with weighted frequency bands and logarithmic scaling applied. **(d)** The function D_x. The red line indicates the threshold that will be applied. **(e)** The novelty curve T_x.

	our approach		**Grosche et al.**	
Filename	Precision	Recall	Precision	Recall
cello1	0.333	0.164	0.730	0.885
clarinet1	0.200	0.114	0.061	0.086
classic2	0.429	0.079	0.388	0.868
classic3	0.000	0.000	0.023	0.500
distguit1	0.857	0.316	0.875	0.737
guitar2	0.880	0.611	0.842	0.889
guitar3	0.976	0.732	0.966	1.000
jazz2	0.692	0.383	0.821	0.681
jazz3	0.838	0.585	0.975	0.736
piano1	1.000	0.632	0.950	1.000
pop1	0.941	0.593	0.719	0.852
rock1	0.875	0.237	0.935	0.729
sax1	0.714	0.556	0.136	0.333
synthbass1	0.846	0.478	0.905	0.826
techno2	0.972	0.625	0.940	0.839
trumpet1	0.538	0.125	0.959	0.839
violin2	0.750	0.288	0.771	0.877
Average	**0.697**	**0.383**	**0.706**	**0.746**

Table 6.1. Precision/Recall Results. An onset was counted as detected correctly if it lied in a tolerance region of 50 ms around the reference onset. Database and reference annotations taken from [29].

softer onsets which are not of interest to us since such onsets do not cause stuttering artifacts in WSOLA. Detecting too many erroneous transients (either positions where no onset takes place or the present onset is not artifact causing) can even degenerate the output audio quality. The transient preservation is realized by a manipulation of the time-stretch function (see Section 6.3). A huge amount of erroneously detected transients therefore leads to a degeneration of the time-stretch function or even to a suppression of the correctly detected transients.

We tested our transient detection method on an onset-annotated dataset of 17 short musical excerpts [29] together with a state of the art onset detection algorithm developed by Grosche et al. [20]. A detected onset was classified as correctly detected, or true positive (TP), in case it was located in a 50 ms tolerance region around a reference onset. Otherwise it was classified as false positive (FP). In case no onset was detected for a reference onset this was considered a false negative (FN). We then computed precision and recall as follows.

$$\text{Precision} = \frac{TP}{TP + FP}, \ \text{Recall} = \frac{TP}{TP + FN} \quad . \tag{6.7}$$

The results can be seen in Table 6.1. It shows that the precision of our method is only slightly worse than the precision of the state of the art method. The recall on the other hand is of course lower. Nevertheless, listening tests revealed that most of the missed transients were not artifact causing. Figure 6.4 shows the detected onsets of both methods for an example excerpt.

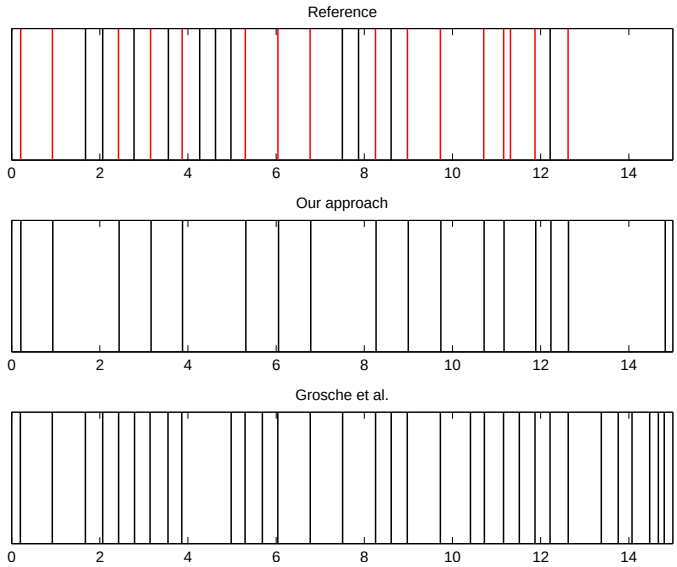

Figure 6.4. Onset detection results for the musical excerpt *pop1* from our dataset. The time is given in seconds. Vertical lines indicate the onset positions. In the reference we indicated onsets that cause stuttering artifacts by red bars and onsets that are unproblematic when time-scale modifying with WSOLA by blue bars. The classification was done by listening tests. The excerpt was time-scale modified using WSOLA with a global time-stretch factor of 2. Afterwards the artifact causing transient were identified manually. One can see that our approach is capable of detecting a large amount of artifact causing transients and at the same time yield only few false positives.

6.3 Transient Preservation

Knowing the positions of the transients in the input audio signal x we now can preserve these transients when time-scale modifying x. Recall that given to our WSOLA algorithm is the input signal x together with a set of anchor points that describe the intended time-stretch function τ. The core idea of the transient preservation is to modify the set of anchor points, and therefore the time-stretch function τ, such that at transient positions we have a local time-stretch factor of 1, i.e. no time-scale modification. Let $p \in [1 : length(x)]$ be the position of a transient in the input signal. To preserve this transient we insert two new anchor points (n_1, m_1) and (n_2, m_2) into the set of anchor points. For these new anchor points it holds that

$$\begin{array}{ll} (i) & n_1 = p - \epsilon_1 \\ (ii) & m_1 = \tau(p) - \epsilon_1 \\ (iii) & n_2 = p + \epsilon_2 \\ (iv) & m_2 = \tau(p) + \epsilon_2 \\ (v) & \frac{n_2 - n_1}{m_2 - m_1} = 1 \end{array} \tag{6.8}$$

with $\epsilon_1, \epsilon_2 \in \mathbb{N}$ (see Figure 6.5). We call the interval $[n1 : n2]$ the *transient window* and denote it by Ξ. Intuitively Ξ marks an area around the transient in the input signal. By using this technique we can manipulate the time-stretch factor locally but at the same time

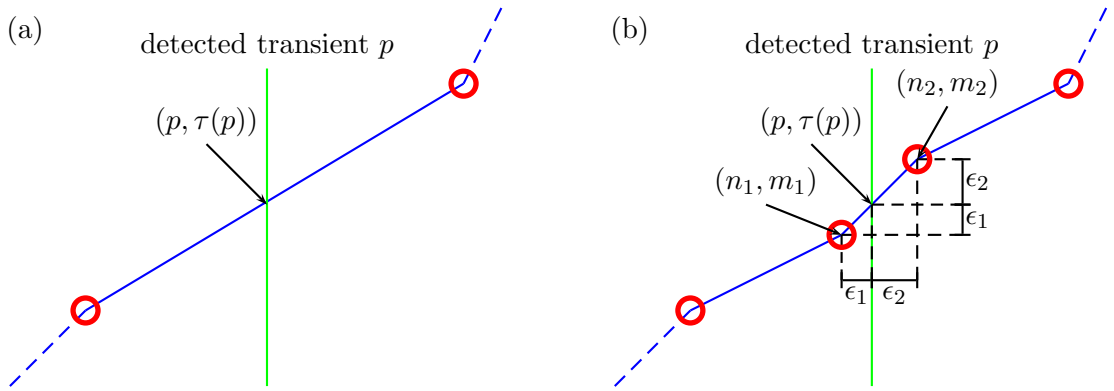

Figure 6.5. (a) A segment of a time-stretch function τ. The anchor points are indicated by red circles. The position p marks the point in the input audio where a transient was detected. **(b)** To preserve the transient at position p two new anchor points (n_1, m_1) and (n_2, m_2) are introduced to create a short segment with local time-stretch factor 1 where $n_1 = p - \epsilon_1$, $m_1 = \tau(p) - \epsilon_1$, $n_2 = p + \epsilon_2$ and $m_2 = \tau(p) + \epsilon_2$.

keeping the global time-stretch function intact. Unfortunately, since WSOLA works with windows of fixed length, it can not guarantee that time-relations specified by the set of anchor points are realized exactly. Just because there are two anchor points that indicate a very small time-scale modification free segment does not mean that there will actually be a time-scale modification free segment in the output signal of WSOLA. To ensure the existence of such a segment around the transient we have to take care that the two new anchor points are not spaced too close to each other. For the sake of simplicity let us first consider the basic OLA scenario neglecting the existence of the window position tolerance Δ_{max}. In this scenario we have to ensure that there are at least $\lceil \frac{1}{1-o} \rceil$ input window positions placed in the interval Ξ where o is the overlap factor of the OLA algorithm. This is because in a signal that was constructed by adding up a sequence of windowed audio segments overlapping by a constant factor o, there are always at most $\lceil \frac{1}{1-o} \rceil$ windowed audio segments that contribute to a given point in the signal. Furthermore, we know that the windows that will be placed in the interval Ξ will all be equidistantly spaced apart by the standard window offset η_o^w. This is because the local slope of the time-stretch function τ is exactly 1 in the interval Ξ by design of the two new inserted anchor points. A slope of 1 indicates that the windows in the input are spaced exactly as the windows in the output, and the windows in the output are all equidistantly spaced apart by η_o^w by design of OLA. Summarizing we have to ensure that there are at least $\lceil \frac{1}{1-o} \rceil$ input window positions spaced apart by η_o^w lying in the interval Ξ to guarantee the existence of an interval $I \subseteq \Xi$ in the input signal such that this interval is not time-scale modified when applying OLA. See Figure 6.6 for an example. Note that since the exact positions of the input windows are not only determined by the time-stretch function τ but also by the fixed output position vector γ, the exact position of I in Ξ can not be precisely specified upfront. From these requirements we now can compute the minimal distance between n_1 and n_2 and therefore the size of the transient window Ξ.

$$length(\Xi) = n_2 - n_1 \geq \left\lceil \frac{1}{1-o} \right\rceil \cdot \eta_o^w \geq \frac{1}{1-o} \cdot \eta_o^w = \frac{1}{1-o} \cdot (1-o) \cdot w_\ell = w_\ell \quad . \tag{6.9}$$

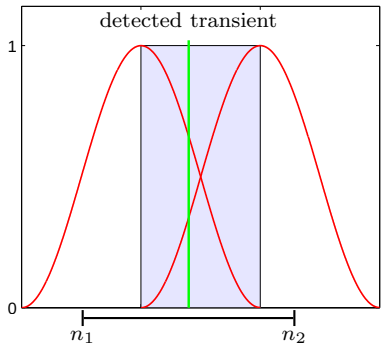

Figure 6.6. In this example we set $o = \frac{1}{2}$. We therefore need at least $\frac{1}{1-\frac{1}{2}} = 2$ input window positions spaced apart by $\eta_o^w = (1-\frac{1}{2}) \cdot w_l = \frac{1}{2} \cdot w_l$ in the interval $\Xi = [n_1 : n_2]$ around the detected transient such that there exists an interval $I \subseteq \Xi$ (blue area) that is not time-scale modified in the output.

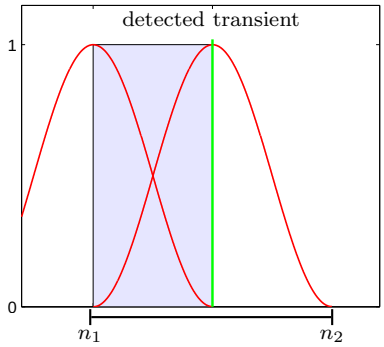

Figure 6.7. An example where an interval $I \subseteq \Xi$ exists but the detected transient does not lie in I (blue area) and is therefore not preserved.

Equation 6.9 yields a lower bound on the size of the interval Ξ to ensure the existence of I. Furthermore, the length of the time-scale modification free interval I can be assessed by

$$length(I) \geq \left(\left\lfloor \frac{n_2 - n_1}{\eta_o^w} \right\rfloor - 1 \right) \cdot \eta_o^w \tag{6.10}$$

since it holds that N windows that are all equidistantly spaced apart by M yield a segment of length $(N-1) \cdot M$ to which only these N windows contribute to. Further, the term $\lfloor \frac{L}{M} \rfloor$ describes the minimal number of points that lie in a segment of length L in case the points are equidistantly spaced apart by M.

At this point we know how large we have to choose Ξ to guarantee the existence of I. Furthermore, we can even estimate the size of I. But we have another problem. Even though we can guarantee the existence of the interval I, we do not know its exact position in the interval Ξ. It is therefore possible that $p \notin I$ for p being the position of the detected transient. See Figure 6.7 for an example. To ensure that a detected transient is always

preserved we have to take care that the time-scale modification free interval I is large enough and will definitely include p. Therefore we will now construct constraints such that fulfilling these constraints guarantees the preservation of the transient.

Recall at this point that we only discussed a transient preservation for the basic OLA technique so far. In WSOLA all window positions may shift by the window position tolerance Δ_{max}. Therefore the whole time-scale modification free segment I may shift by the same amount. In WSOLA, the subset relationship $I \subseteq \Xi = [n_1 : n_2]$ therefore has to be replaced by $I \subset [n_1 - \Delta_{max} : n_2 + \Delta max]$. Note that all other assessments stay valid. To ensure that a detected transient at position p lies in the interval I it therefore has to hold that

$$p \in Q = [n_2 + \Delta_{max} - length(I) : n_1 - \Delta_{max} + length(I)] \quad . \tag{6.11}$$

The interval Q includes all points q such that $q \in I$ is independent of the position of I in $[n_1 - \Delta_{max} : n_2 + \Delta_{max}]$. We therefore can conclude

$$\begin{aligned} & p \in Q = [n_2 + \Delta_{max} - length(I) : n_1 - \Delta_{max} + length(I)] \\ \Leftrightarrow \quad & p \in [p + \epsilon_2 + \Delta_{max} - length(I) : p - \epsilon_1 - \Delta max + length(I)] \\ \Leftrightarrow \quad & p \geq p + \epsilon_2 + \Delta_{max} - length(I) \text{ and } p \leq p - \epsilon_1 - \Delta_{max} + length(I) \\ \Leftrightarrow \quad & \epsilon_2 \leq length(I) - \Delta_{max} \text{ and } \epsilon_1 \leq length(I) - \Delta_{max} \end{aligned} \tag{6.12}$$

We therefore get the following set of constraints to guarantee the preservation of the transient

$$\begin{aligned} & (i) \quad length(\Xi) = length([n_1 : n_2]) = \epsilon_1 + \epsilon_2 \geq w_\ell && (6.8), (6.9) \\ & (ii) \quad \epsilon_1 \leq length(I) - \Delta_{max} && (6.12) \\ & (iii) \quad \epsilon_2 \leq length(I) - \Delta_{max} && (6.12) \end{aligned} \tag{6.13}$$

But since we do not know the exact size of I we use the lower bound on I from Equation 6.10 to strengthen the constraints.

$$\begin{aligned} & (i) \quad length(\Xi) = length([n_1 : n_2]) = \epsilon_1 + \epsilon_2 \geq w_\ell && (6.8), (6.9) \\ & (ii) \quad \epsilon_1 \leq (\lfloor \tfrac{length(\Xi)}{\eta_o^w} \rfloor - 1) \cdot \eta_o^w - \Delta_{max} && (6.10) \\ & (iii) \quad \epsilon_2 \leq (\lfloor \tfrac{length(\Xi)}{\eta_o^w} \rfloor - 1) \cdot \eta_o^w - \Delta_{max} && (6.10) \end{aligned} \tag{6.14}$$

6.4 Limitations of the Transient Preservation

To ensure that the segment that is defined by the two new anchor points (n_1, m_1) and (n_2, m_2) is indeed time-scale modification free it makes sense to delete any other anchor points (n, m) with $n_1 < n < n_2$ or $m_1 < m < m_2$. This can lead to problems in case two transients are close together and an anchor point, that were inserted to preserve the other transient, is deleted. We therefore proceed in an iterative fashion and choose the next transient that should be preserved by picking the highest peak in the novelty curve

T_x. After having modified the set of anchor points to preserve the just found transient, the indicating peak is removed from the novelty curve and the next highest peak is picked. This process makes sense since the hight of a peak correlates directly to the energy, and therefore to the loudness of the causing transient and of course stuttering artifacts at a loud transient are more noticeable than the artifacts at a rather unobtrusive transient. To save anchor points that were inserted to preserve a transient from being deleted during the process we separate the set of anchor points in two parts. Let A be the set of anchor points. We define

$$A = A_{deletable} \cup A_{nondeletable} \quad , \tag{6.15}$$

where the sets $A_{deletable}$ and $A_{nondeletable}$ are disjoint. At the beginning of the transient preservation process we define $A_{deletable}$ to be the set of all anchor points that describe τ and $A_{nondeletable} = \emptyset$. Whenever we now want to insert new anchor points (n_1, m_1) and (n_2, m_2) into the set of anchor points and there exists an anchor point $(n', m') \in A_{nondeletable}$ with $n_1 < n' < n_2$ or $m_1 < m' < m_2$ the current transient is skipped and we proceed with the next transient. Otherwise the new anchor points are added to $A_{nondeletable}$ and we have $A_{nondeletable} = A_{nondeletable} \cup \{(n_1, m_1), (n_2, m_2)\}$. In case there exists an anchor point $(n'', m'') \in A_{deletable}$ with $n_1 < n'' < n_2$ or $m_1 < m'' < m_2$ this anchor point is deleted and we have $A_{deletable} = A_{deletable} \backslash (n'', m'')$. This prevents the degeneration of the time-stretch function τ and ensures that the strongest transients are preferred during the process. Figure 6.8 shows an example. The procedure is summerized in Algorithm 4.

Using this technique ensures that the stronger transients are preferred over weaker transients. But at the same time it implies limitations on the number of transients that can be preserved in an audio signal of a certain length. More precisely, for a given local time-stretch factor ψ and a fixed length of transient windows Ξ_ℓ there exists a minimal distance δ_{min} between two transient positions such that both transients can be preserved. This distance can be computed as follows

$$\delta_{min} = \begin{cases} \Xi_\ell + 1 & \text{if } \psi \geq 1 \\ \frac{1}{\psi} \cdot \Xi_\ell + 1 & \text{if } \psi < 1 \end{cases} \tag{6.16}$$

The reason for this is the following: Assume we have two transients at positions p' and p'' in our input signal and assume further that w.l.o.g. $p' < p''$. To preserve those transients we want to insert new anchor points (n'_1, m'_1), (n'_2, m'_2) for p' and (n''_1, m''_1), (n''_2, m''_2) for p'' into the set of anchor points such that it holds that

$$\begin{array}{ll} (i) & n'_1 = p' - \epsilon_1 \\ (ii) & m'_1 = \tau(p') - \epsilon_1 \\ (iii) & n'_2 = p' + \epsilon_2 \\ (iv) & m'_2 = \tau(p') + \epsilon_2 \\ (v) & \frac{n'_2 - n'_1}{m'_2 - m'_1} = 1 \end{array} \tag{6.17}$$

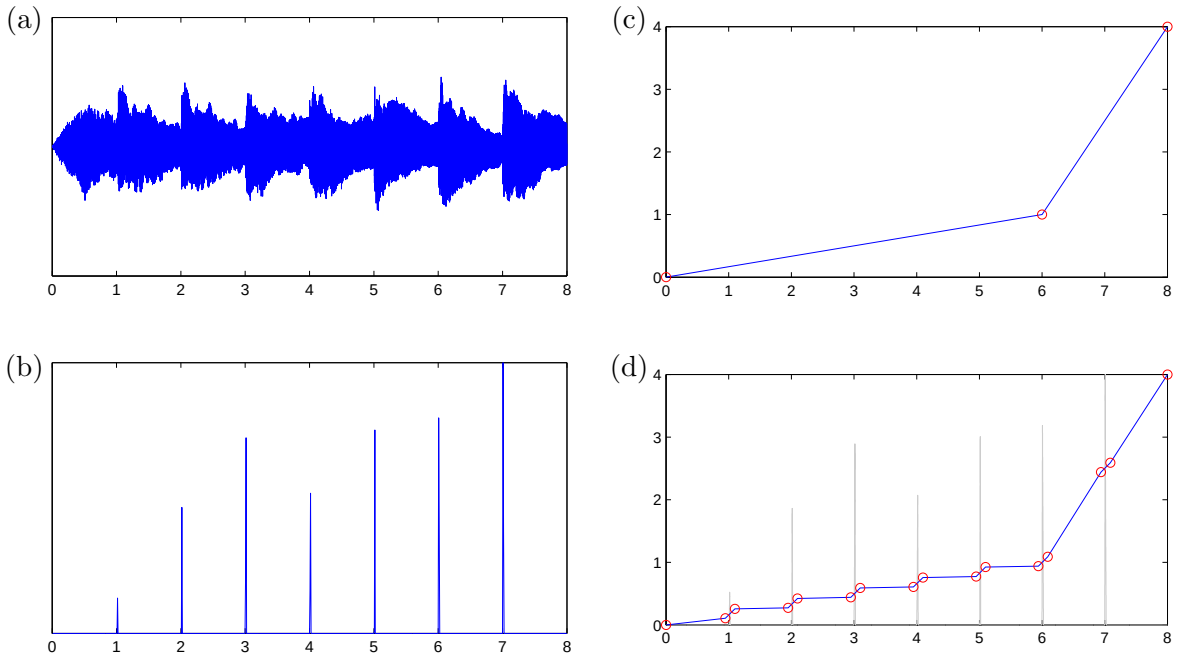

Figure 6.8. **(a)**. The input signal x. **(b)** The novelty curve T_x. **(c)** The time-stretch function τ represented by a set of anchor points (red circles). **(d)** The modified time-stretch function. For each peak in T_x two new anchor points were introduced. For the sixth peak the anchor point lying in between the two new anchor points had to be deleted to ensure a local time-stretch factor of 1. Note that the novelty curve T_x was plotted in the background for demonstration purposes.

Algorithm 4: Transient Preservation

Data: DT signal x, set of anchor points A that describes τ, ϵ_1, ϵ_2.
Result: Modified set of anchor points A such that transients in x are preserved when time-scale modifying it.

begin

/* Compute the novelty curve T_x */
$T_x \longleftarrow ComputeNoveltyCurve(x)$;

/* Split the set of anchor points */
$A_{deletable} \longleftarrow A$;
$A_{nondeletable} \longleftarrow \emptyset$

while $T_x \neq 0$ **do**

/* Pick the highest peak in T_x */
$p \longleftarrow indexOfMaxValue(T_x)$;

/* Compute new anchor points */
$(n_1, m_1) \longleftarrow (p - \epsilon_1, \tau(p) - \epsilon_1)$;
$(n_2, m_2) \longleftarrow (p + \epsilon_2, \tau(p) + \epsilon_2)$;

/* Proceed only if no non deletable anchor point lies between the two new anchor points */
if $\neg\exists(n', m') \in A_{nondeletable} : n_1 < n' < n_2 \vee m_1 < m' < m_2$ **then**

/* If a deletable anchor point lies between new anchor points delete it */
if $\exists(n'', m'') \in A_{deletable} : n_1 < n'' < n_2 \vee m_1 < m'' < m_2$ **then**
$A_{deletable} \longleftarrow A_{deletable} \backslash (n'', m'')$;

/* Insert the two new anchor points into the set of non deletable anchor points */
$A_{nondeletable} \longleftarrow A_{nondeletable} \cup \{(n_1, m_1), (n_2, m_2)\}$;

/* Remove the just processed peak from the novelty curve */
$T_x(p) \longleftarrow 0$;

/* Merge the sets of deletable and non deletable anchor points */
$A \longleftarrow A_{deletable} \cup A_{nondeletable}$;

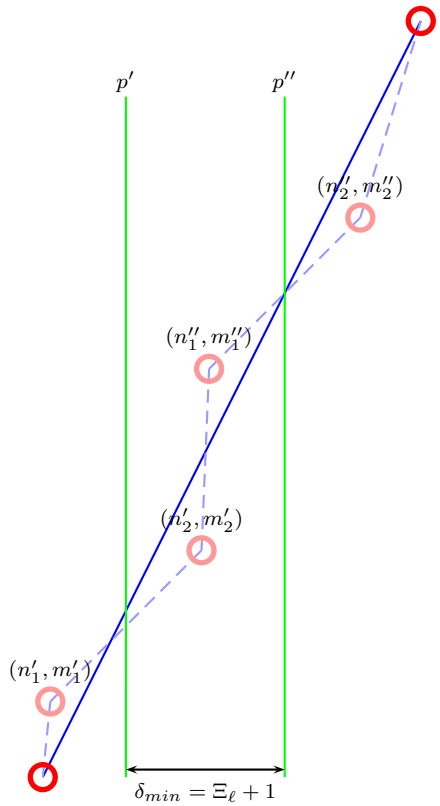

Figure 6.9. The minimal distance δ_{min} between transient positions p' and p'' such that both transients can be preserved in case the local time-stretch factor ψ is greater one. The original time-stretch function τ is indicated by the straight blue line while the time-stretch function resulting from the insertion of the new anchor points is indicated by the dashed blue line.

and accordingly for n_1'', m_1'', n_2'', m_2'' and p''. To circumvent that one of the two transients is skipped during the transient preservation process it has to hold that

$$\begin{array}{lllll} (i) & n_2' < n_1'' & \Leftrightarrow & p' + \epsilon_2 < p'' - \epsilon_1 & \Leftrightarrow & \Xi_\ell < p'' - p' \\ (ii) & m_2' < m_1'' & \Leftrightarrow & \tau(p') + \epsilon_2 < \tau(p'') - \epsilon_1 & \Leftrightarrow & \Xi_\ell < \tau(p'') - \tau(p') \end{array} \tag{6.18}$$

Note that the difference $\tau(p'') - \tau(p')$ can be computed by $\psi \cdot (p'' - p')$ since we assume a constant time-stretch factor and the time stretch function τ is therefore linear. In case the local time-stretch factor ψ is greater or equal to one we therefore automatically fulfill the second requirement in Equation 6.18 when fulfilling the first, and the first requirement is fulfilled when $p'' - p' \geq \Xi_\ell + 1$. In contrary, in case the local time-stretch factor ψ is smaller than one, we fulfill the first requirement automatically whenever fulfilling the second which is fulfilled when $p'' - p' \geq \frac{1}{\psi} \cdot \Xi_\ell + 1$. Examples can be seen in Figure 6.9 and Figure 6.10.

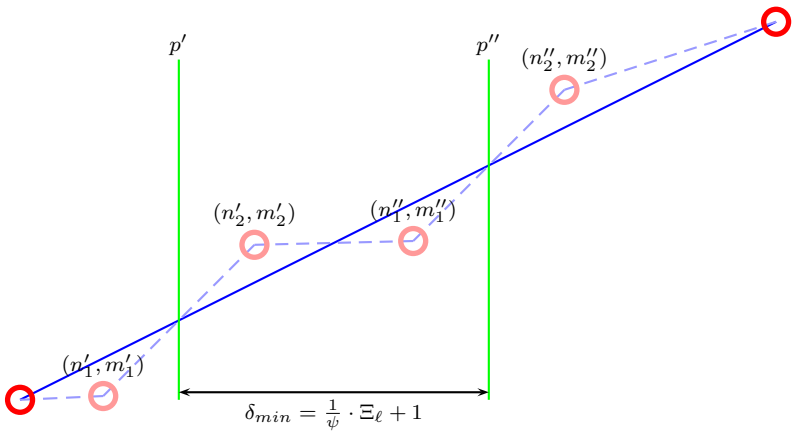

Figure 6.10. The minimal distance δ_{min} between transient positions p' and p'' such that both transients can be preserved in case the local time-stretch factor ψ is smaller one. The original time-stretch function τ is indicated by the straight blue line while the time-stretch function resulting from the insertion of the new anchor points is indicated by the dashed blue line.

Chapter 7

Listening Test

To gain further insights into the quality of WSOLA and the transient preservation in WSOLA we performed a listening test. While performing this test we had the following leading questions:

- How does WSOLA perform on an absolute scale? (In comparison to a *perfect* time-scale modification algorithm)
- How does WSOLA perform in comparison to other time-scale modification algorithms?
- How great is the effect of transient preservation in WSOLA?

In Section 7.1 we describe the dataset we performed the listening test on. The test setup is discussed in 7.2 and results are presented in 7.3.

7.1 Test Dataset

Our test dataset consists of 10 short sonified MIDI files, 4 extracted from the RWC database [17] and 6 created by ourself (see Table 7.1). The files were chosen to cover music of different styles, instrumentation, tempo and complexity. Furthermore, our self-created files were designed to have a repetitive structure such that possible artifacts occur several times and therefore can be better investigated by the listening test participants. All files were sonified using a state-of-the-art MIDI synthesizer.

7.2 Test Setup

From each sonified MIDI file in our test dataset we produced 4 time-scale modified versions with a global time-stretch factor of two, using different time-scale modification algorithms:

Name	content	additional comment	length in seconds
BASS1	Electric bass.		7.5
BONGO1	Regular beat played on Bongos.		6.2
DRUM1	Rock beat played on a standard drum-set.	Open Hi-Hat and room reverb.	7.4
FLUTE1	Flute playing a simple melody.	Strong natural tremolo at a long note.	6.9
GENRE1	Rock music. Playing instruments: flute, distorted electrical guitar, bass, drums	Polyphone music. Taken from RWC database.	10.6
JAZZ1	Piano (left and right hand)	Strong agogics. Taken from RWC database.	13
OWN1	Electronic music. Bass and melody synthesizer	The melody synthesizer produces sharp, noise-like onsets.	14.7
POP3	Pop music. Playing instruments: flute, piano, bass, steel guitar, quiet drums	Taken from RWC database.	11.1
POP4	Pop-Rock music. Playing instruments: distorted guitar, synthesizer, bass, drums.	Drums play fill-in at the end. Taken from RWC database.	9
VIOLIN1	Two violins.	Strong vibrato.	10.5

Table 7.1. Short description of the used musical excerpts.

- A free MATLAB implementation of the Phase Vocoder[1] similar to the implementation explained in Section 5.4.
- The commercial MPEX4 algorithm[2].
- Our implementation of WSOLA.
- Our implementation of the transient preserving WSOLA (TP-WSOLA).

The parameters and settings of the algorithms can be found in Table 7.2.

Since the onset detection itself was not part of the evaluation, we decided to eliminate possible error sources by extracting all onset positions in the audio signals from the MIDI files and therefore skipping the possibly erroneous transient detection step in TP-WSOLA. Additional to the four time-scale modified versions we also produced a sonification of each MIDI file that had half the tempo of the original version. Since time-scale modification algorithms are supposed to produce modifications of audio signals that sound as if the recorded musical piece was initially performed on a different tempo, the half-tempo MIDI sonification can be seen as the best possible result a time-scale modification algorithm could produce. We decided to present this version along with the time-scale modified versions to the test participants to give them a reference on how the perfect result should sound. Nevertheless, the half-tempo MIDI reference was not specifically marked as such.

For all the excerpts from our test set the listening test proceeded as follows: The original audio recording along with the five time-scaled versions (four algorithm outputs plus the half-tempo MIDI reference) in randomized order were presented to the test participants

[1] `http://labrosa.ee.columbia.edu/matlab/pvoc/`
[2] `http://mpex.prosoniq.com/`

Phase Vocoder:	Analysis window size:	1024 samples (Hann-window)
	Hop factor ϱ_{hop}:	256 samples
	Synthesis window size:	1024 samples (Hann-window)
WSOLA:	Window size w_ℓ:	552 samples (Hann-Window)
	Overlap factor o:	0.5
	Tolerance Δ_{max}:	276 samples
Transient preserving WSOLA:	Window size w_ℓ:	552 samples (Hann-Window)
	Overlap factor o:	0.5
	Tolerance Δ_{max}:	276 samples
	Length of transient window Ξ:	1104 samples
	ϵ_1:	552 samples
	ϵ_2:	552 samples
MPEX4:	The algorithm was run in high quality mode (polyphone complex).	

Table 7.2. The parameters and settings for the applied algorithms. All audio signals were sampled at a rate of 22050 Hz.

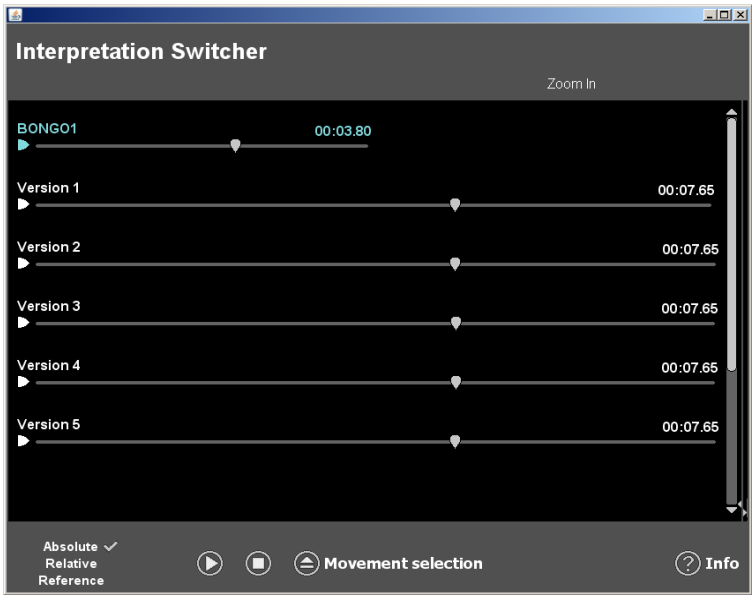

Figure 7.1. Screenshot of the InterpretationSwitcher.

using the InterpretationSwitcher [34] (see Figure 7.1). They were asked to rate the quality of each time-scale modified version on a scale from 5 (excellent) to 1 (bad) while knowing that among the five versions there always is one version that can be considered *perfect* and therefore earns the best possible grade. We decided to give this information to the test participants since even the perfect half tempo MIDI reference might sound very unnatural just because of the huge temporal distortion. The questionnaire used for the listening test can be found in Appendix B.

Overall ten people participated in the listening test. We are aware of the fact that this small number of probands is hardly enough to get significant results. On the other hand, we believe that it is enough to get a general feeling of the quality of the different time-scale modification algorithms.

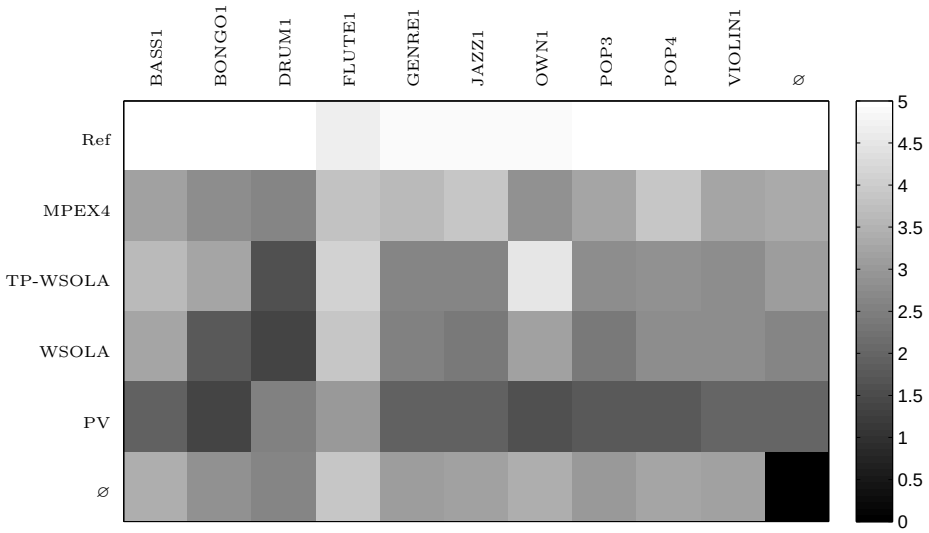

Figure 7.2. Overview of the results of the listening Test. The mean opinion score is color-coded for each excerpt from our dataset and each time-scale modified version of it. Additionally, the average values for each excerpt over all time-scale modification techniques were computed, yielding an indicator for the *complexity* of the excerpt (bottom row), as well as the average values for each time-scale modification technique over all excerpts, indicating the overall quality of the respective time-scale modification technique (last column).

7.3 Results

For all audio recordings in our test dataset we computed the mean opinion score of the results for each of the five time-scale modified versions. The results can be seen in Figure 7.2 and Figure 7.3. Comments that were given by the test participants during the listening test are listed in Appendix C.

The listening test therefore yielded the following results:

How does WSOLA perform on an absolute scale? The small number of test participants makes it hard to answer this question appropriately. What we can state is that for two examples (FLUTE1 and OWN1) TP-WSOLA algorithm scored almost as good as the half-tempo MIDI sonification and can therefore be considered to produce results close to perfection. This might sound promising, but having a closer look at these two examples relativizes the results: The FLUTE1 example seems to be an easy example in general, considering that all four algorithms scored relatively good here. The flute onsets are soft and therefore not likely to cause artifacts during the time-scale modification process. The test participants also noticed the half-tempo tremolo of the flute and described it as *vibrating* but not as disturbing. The OWN1 example on the other hand is predestinated to score good when being time-scale modified with the TP-WSOLA algorithm: Its bass line is very monotone and simple while the melody synthesizer produces very short tones that can be restored perfectly by the transient preservation. An other noticeable example from the test dataset is the DRUM1 example. Here, the WSOLA algorithm as well as the TP-WSOLA algorithm score particularly bad. This is because the stuttering artifacts

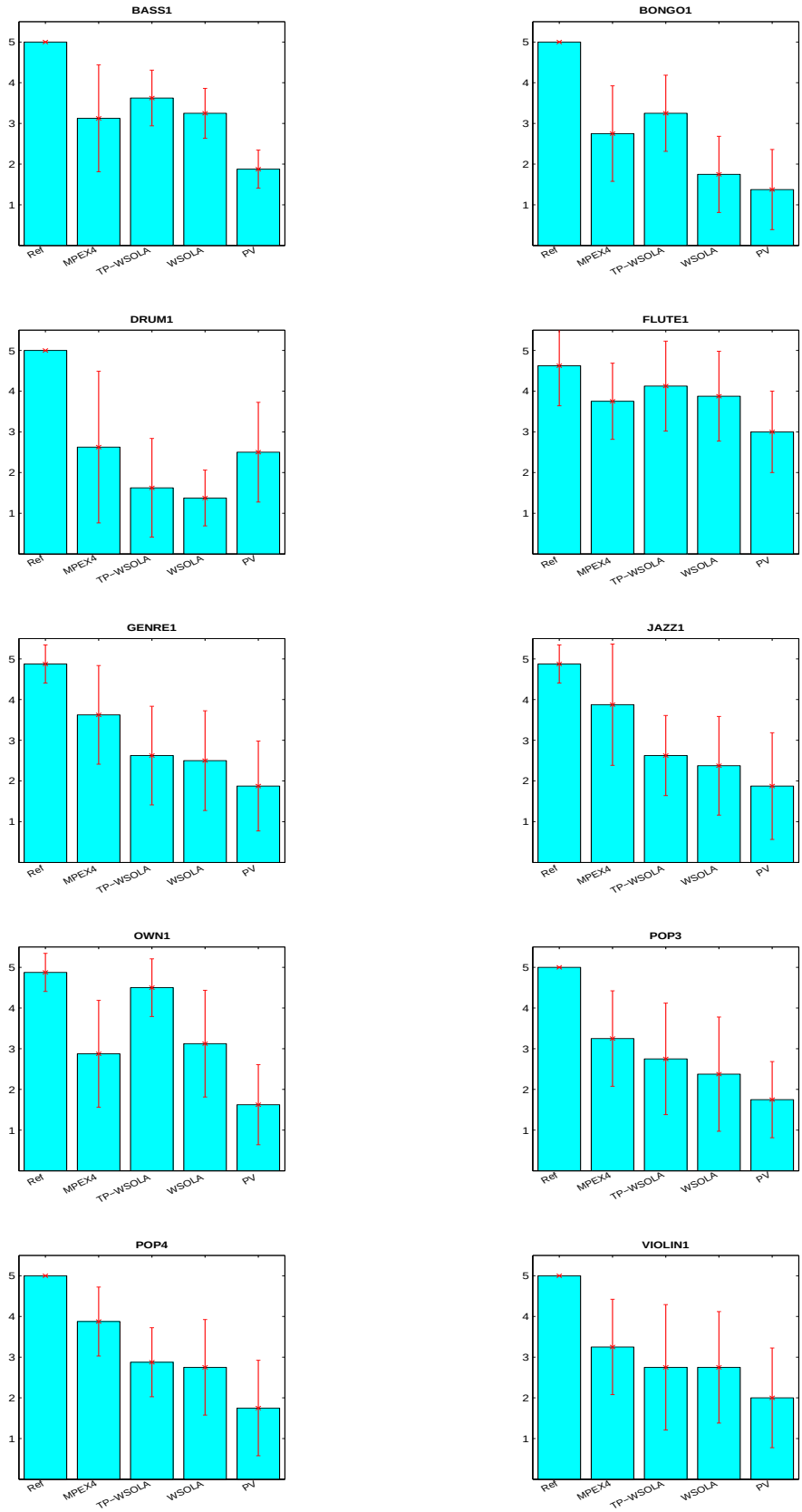

Figure 7.3. Results of the listening test for the half-tempo MIDI reference (Ref), the MPEX4 algorithm, TP-WSOLA, WSOLA and the Phase Vocoder (PV). The standard deviation is indicated by red bars.

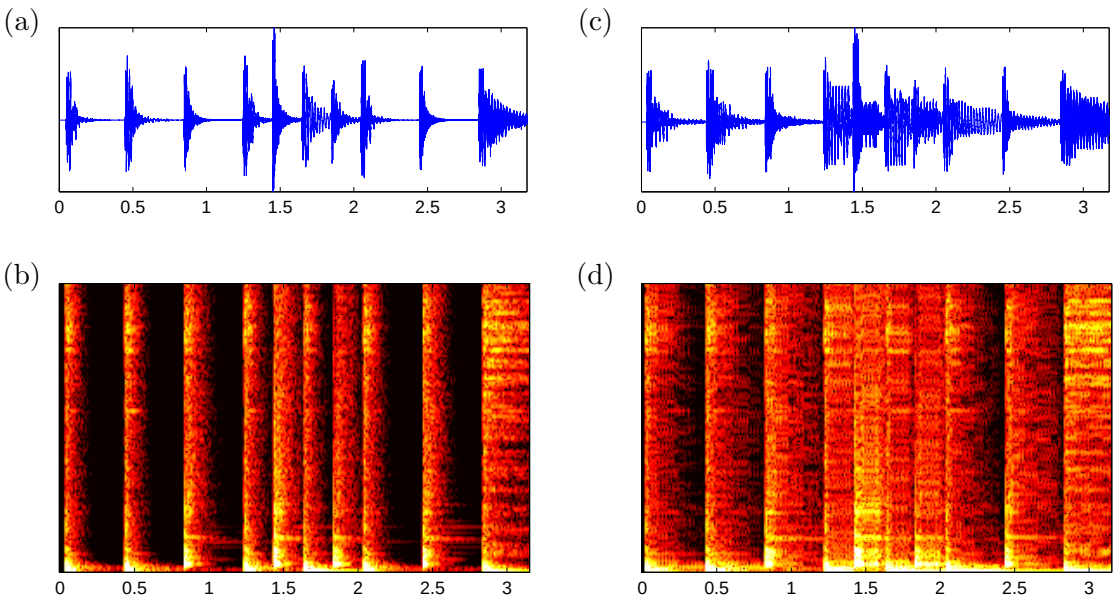

Figure 7.4. **(a)** The waveform of the first 3 seconds from the half-tempo MIDI reference of DRUM1. **(b)** The spectrogram related to (a). Note the rather long decay phases of the notes resulting from room reverb. **(c)** The waveform of the first 3 seconds from DRUM1 stretched with a time-scale factor of 2 using TP-WSOLA. **(d)** The spectrogram related to (c). Note that a lot of stuttering artifacts are visible in the decay phases of the notes even though the actual onsets are preserved by the algorithm.

cause an *electric* and *metallic* sound of the drums. The stuttering artifacts also occur when using TP-WSOLA because of the room reverb in this example (See Figure 7.4). Summarizing we can say that the quality of time-scale modified audio signals produced with WSOLA strongly depend on the input audio signal.

How does WSOLA perform in comparison to other time-scale modification algorithms? For most of the examples in our test dataset WSOLA and TP-WSOLA algorithm perform slightly worse than the commercial MPEX4 algorithm and slightly better than the Phase Vocoder. But unlike MPEX4, which robustly yields rather good results for almost any example, the quality of results for WSOLA is much more signal dependent.
In this context, it is also important to point out that the commercial MPEX4 and our MATLAB implementation of WSOLA differ significantly concerning the running time: while our WSOLA implementation takes about one to two seconds to compute the time-scale modification of an audio signal of roughly five seconds with a constant time-stretch factor of two, the MPEX4 algorithm in its highest quality mode (in which we run the algorithm for our listening test) needs about 20 seconds.

How great is the effect of transient preservation in WSOLA? We can observe that on average TP-WSOLA scores always better or equal to the standard WSOLA. In

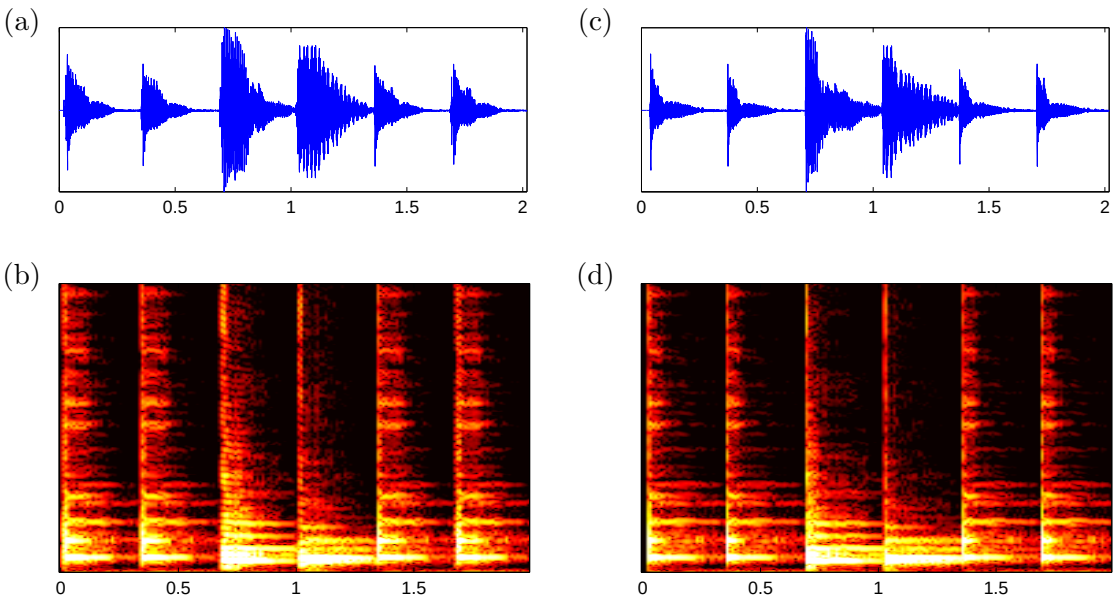

Figure 7.5. **(a)** The waveform of the first 2 seconds from BONGO1 stretched with a time-scale factor of 2 using WSOLA. **(b)** The spectrogram related to (a). Note that the stuttering artifacts are visible very well. **(c)** The waveform of the first 2 seconds from BONGO1 stretched with a time-scale factor of 2 using TP-WSOLA. **(d)** The spectrogram related to (c). One can observe that the transient preservation reduces stuttering artifacts at transients significantly. This can be seen in the waveform (steep attack phase and steady decay phase with transient preservation) as well as in the spectrogram (sharper transient regions in the spectrogram).

some cases it even causes huge quality improvements. Examples for this improvements are BONGO1 and OWN1, where TP-WSOLA scores significantly better than the standard WSOLA. Again this is mainly due to the structure of those audio signals. The short bongo hits in BONGO1 as well as the short noise-like bursts of the melody synthesizer in OWN1 can be restored perfectly by the transient preservation during the time-scale modification. On the other hand, these bursts also cause major stuttering artifacts when time-scale modifying with the standard WSOLA technique (see Figure 7.5). For other instruments like for example the bass in BASS1 the stuttering artifacts are not that well perceivable and the score difference is therefore not that high. For examples like VIOLIN1, where very little note onsets occur and the onsets are also rather soft there is virtually no difference between the standard WSOLA and its transient preserving version.

Chapter 8

The Connector's Pipeline

The purpose of the Connector is to produce a transition from a given audio recording to another audio recording in a database that fulfills given constraints. This transition should be as pleasant as possible to the ear of the listener. Of course, *pleasant* is a very vague term in this context. There are extremely many aspects of the audio signals that could count towards the quality of a produced transition like harmony, beat progression, genre, timbre, volume or even just the quality of the audio recordings that are used. Furthermore, it is hard to argue about what sounds pleasant to everybody since the same transition might sound differently appealing to different people. The Connector therefore focuses on those aspects that are, at least intuitively, generally important when it comes to the euphony of transitions which are the harmonic progression and the beat. In this chapter we describe the Connector in its current state.

Given to the Connector are an audio recording for which a *transition region* is specified along with a set of user defined constraints. The transition region marks the part in the audio recording where the transition to another audio recording should take place. Like shown in Figure 1.3, the Connector works in three stages. In the matching stage, the audio in the transition region is converted into a suitable representation that allows to find other audio recordings with regions that share a similar harmonic progression in a database. Picking the best match that fulfills all constraints specified by the user gives us the audio recording that we will bridge to. The region in this audio recording that is similar in its harmonic progression to the transition region is referred to as the *matched region*. At this point, the warping stage of the Connector starts. The goal of this stage is to rhythmically synchronize the transition region with the matched region. This is done by applying TSM algorithms to both regions such that afterwards the beat positions of both regions are temporally aligned. The idea of synchronizing the beats while bridging from one audio recording to another originates from the field of DJing. Since the beat is a very important element in danceable music, it allows the DJ to create extremely long audio recordings that are danceable from the beginning to the end without breaks (see for example [3]). This continuity is exactly what we want to maintain during a transition. In the last stage of the Connector the two audio recordings are blended by overlaying the transition region and the matched region and applying a simple crossfade. Since both regions are now similar in their harmonic progression and rhythmically synchronized, the

transition should sound euphonious.

This chapter is structured as follows. In Section 8.1 we introduce other work in the field of music generation. In Section 8.2 and Section 8.3 we explain the matching process in detail while we discuss the structure of our database in Section 8.4. Finally we have a few remarks on the warping and the blending stage of the CONNECTOR in Sections 8.5 and 8.6.

8.1 Related Work

The idea of connecting existing music by creating euphonious transitions to generate new music is not a recent one. At the end of the 18^{th} century, *Musikalische Würfelspiele* were a popular pastime. In this game a piano player had to compose music by concatenating measures from known pieces that were randomly chosen by the roll of a dice [22, 16]. Nowadays this idea has shifted towards the digital world. In times were large amounts of musical scores are available in the form of MIDI data, an automated music generation system based on these files is proposed in [6]. This approach works in the so called *symbolic domain* which means that it does not deal with existing audio signals but with some abstract representation of music. Music generation in the symbolic domain can be considered easier to be handled than music generation in the audio domain since the abstract representation allows for much more flexibility. Unfortunately to be able to listen to music in the symbolic domain it either needs to be played by human performers or synthesized by a machine. But since human performances are expensive and synthesized music is typically inferior in quality to real world audio recordings, music generation in the symbolic domain is often not an option. But there also exist approaches working in the audio domain. A modern field in which techniques to concatenate audio recordings in an appealing way are very important is the field of DJing [3]. While for a DJ the harmonic similarity of two connected audio recordings usually plays a minor role, a good rhythmical transition is essential. A tool to automate the process of finding good rhythmical transitions between two audio recordings automatically is proposed in [24]. The work of [30] comes closest to our approach. It describes a framework that allows to concatenate a set of audio recordings to a single long audio recording. The audio recordings are ordered such that euphonious transitions between the clips are possible. The positions of the transitions are chosen to maximize the local harmonic and rhythmic similarity of the two audio recordings. The work of [45] aims to synthesize a soundtrack for a given visual data stream from a single audio recording, the so called *example*. By linking events in the visual data stream with specific parts in the example the complete soundtrack can be rendered.

The approach of concatenating audio recordings can even be used to synthesize new sounds rather than complete audio recordings. To this end audio recordings are analyzed and segmented in short *units*. This is done for large collections of audio data. The units can then be used to synthesize target sounds by rearranging them. This technique is called *Concatenative Sound Synthesis* and is described in [38, 39, 40, 41].

8.2 Matching

In the first stage of the CONNECTOR an audio recording should be found in a database that is at some point similar in its harmonic progression to the transition region. We therefore first need an appropriate representation of audio signals that allows for comparing them in terms of harmonic content. The chroma features introduced in Chapter 2 have shown to capture the desired property very well. They abstract away from instrumentation and timbre information and are designed to solely focus on the tonal content of the audio signal. Unfortunately they do not abstract away from timing information since they are computed at a fixed rate. Although this is in general a desirable property, in the context of the CONNECTOR this would make the matching process rather complicated. In the simplest case we can state similarity in harmonic progression if we have two feature sequences of the same length and the features can be considered similar pairwise. But assume we want to compare the feature representation of two interpretations of the same musical piece and assume further that one of the two interpretations was performed at half the tempo of the other. We know that the harmonic progression of both interpretations is exactly the same, but we can not compare the two feature sequences since they are not of the same length. There exist techniques that allow for finding similarities in feature sequences where the compared sequences do not need to be of the same length (Dynamic Time Warping [33]) but these techniques are computationally rather expensive in comparison to the naive approach. Furthermore it has shown that not only the harmonic similarity is important to get euphonious results when blending from one audio recording to another, but also the alignment of the beats in both audio recordings. Therefore it is important that the harmonic similarity still exists after the temporal alignment of the beats in the transition region and the matched region. The solution to this issue are the beat-synchronous chroma features that were also introduced in Chapter 2. Since these features use the musical beat-level as their time axis we can avoid matching techniques like Dynamic Time Warping and use the naive approach of comparing two feature sequences of the same length by pairwise comparison of the features. Furthermore this also solves the problem of keeping the harmonic content of two audio recordings similar after the alignment of their beat positions since by design of the beat-synchronous chroma features two feature sequences are similar exactly in case the two underlaying audio recordings share a similar harmonic progression on the beat level.

Note that to compute the beat-synchronous features it is essential to know all beat positions of the underlaying audio recording. These could for example be extracted by using a beat tracker. Unfortunately the problem of beat tracking is not solved completely so far since the task is rather hard for some kinds of music like for example violin music [21]. Although there already exist a lot of approaches, like for example in [8, 28, 2, 13], there exists no beat tracker to our knowledge that could reliably predict the beat positions for audio recordings of all kinds of music. But since we want to work on a large variety of different musical styles we just avoid this problem for the moment and assume all beat positions for all audio recordings to be given. We therefore use a fully beat annotated database in which all beat positions of all audio recordings are annotated manually.

The matching technique that we use in the CONNECTOR was introduced in [35]. Given a query audio signal, which is the transition region in our case, for which we computed

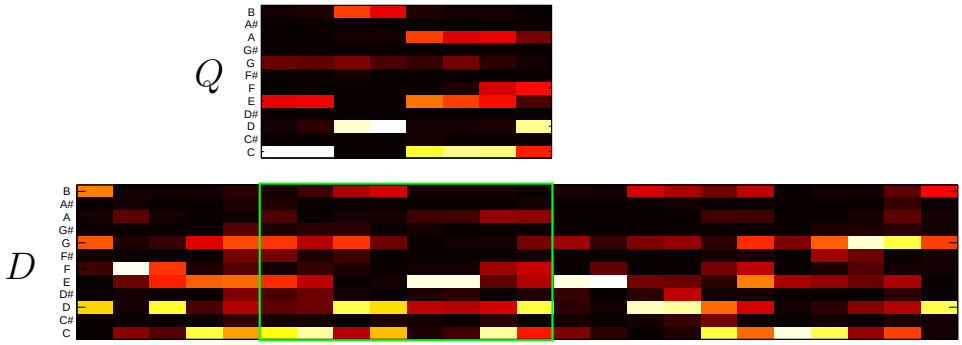

Figure 8.1. Visualization of the task of audio matching. Given a sequence of beat-synchronous chroma vectors Q computed from our query audio signal, we want to find a subsequence of chroma vectors in the database D such that the chroma vectors can be considered similar pairwise. For this example the database was chose to be extremely small for visualization purposes. The subsequence of chroma vectors that is most similar to the query sequence is marked by a green box.

beat-synchronous chrome features, our goal is to find a segment in an audio recording from a database that has a similar harmonic progression to our query audio signal on the beat level. It is therefore important that the database holds beat-synchronous chroma features of all contained audio recordings as well. To this end, we can simply see the database as a concatenation of the beat-synchronous chroma features of all audio recordings that are included in the database. Additionally we keep track of the boundaries in a separate data-structure. It is therefore possible to map a certain chroma vector in the database to a decisive position in a decisive audio recording. We now need to find a sequence of chroma vectors in the database that can be considered similar to the sequence computed from our query audio signal. Figure 8.1 illustrates this task.

The first step towards this goal is to define a distance measure for chroma vectors. Recall that all chroma vectors we work with are normalized. Therefore the inner product of two chroma vectors $\vec{x}$ and $\vec{y}$, which is denoted by $\langle \vec{x}, \vec{y} \rangle$ coincides with the cosine of the angle between $\vec{x}$ and $\vec{y}$ and therefore quantizes the distance of $\vec{x}$ and $\vec{y}$. For this reason we simply define our distance measure δ as

$$\delta(\vec{x}, \vec{y}) = 1 - \langle \vec{x}, \vec{y} \rangle \tag{8.1}$$

Note that $\langle \vec{x}, \vec{y} \rangle \in [0, 1]$ and therefore also $\delta(\vec{x}, \vec{y}) \in [0, 1]$. To extend this distance measure to sequences of features $\vec{X} = (\vec{x}^1, \vec{x}^2, ..., \vec{x}^L)$ and $\vec{Y} = (\vec{y}^1, \vec{y}^2, ..., \vec{y}^L)$ of the same length L we define

$$\delta^L(\vec{X}, \vec{Y}) = \frac{1}{L} \sum_{\ell=1}^{L} \delta(\vec{x}^\ell, \vec{y}^\ell) \tag{8.2}$$

Having the query sequence of chroma vectors $\vec{Q} = (\vec{q}^1, \vec{q}^2, ...\vec{q}^M)$ and the sequence of database chroma vectors $\vec{D} = (\vec{d}^1, \vec{d}^2, ...\vec{d}^N)$ where $M < N$ we can now define the distance

function $\Delta : [1 : n] \to [0, 1]$ by

$$\Delta(i) = \delta^M((\vec{q}^1, \vec{q}^2, ...\vec{q}^M), (\vec{d}^i, \vec{d}^{i+1}, ...\vec{d}^{i+M-1})) \tag{8.3}$$

for $i \in [1 : N - M + 1]$ and

$$\Delta(i) = 1 \tag{8.4}$$

for $i \in [N-M+2 : N]$. The value $\Delta(i)$ describes the distance between the query sequence of chroma vectors and the subsequence of database chroma vectors starting with the i^{th} vector.

Notice that the function Δ can be computed very easily. This is the case since we can compute all interesting values of the distance measure δ by a single matrix multiplication. Recall that we can write

$$\begin{aligned} Q &= (\vec{q}^1, \vec{q}^2, \cdots, \vec{q}^M) = \begin{pmatrix} q_{12,1} & & & q_{12,M} \\ \vdots & & \iddots & \\ q_{2,1} & q_{2,2} & & \\ q_{1,1} & q_{1,2} & \cdots & q_{1,M} \end{pmatrix} \\ D &= (\vec{d}^1, \vec{d}^2, \cdots, \vec{d}^N) = \begin{pmatrix} d_{12,1} & & & d_{12,N} \\ \vdots & & \iddots & \\ d_{2,1} & d_{2,2} & & \\ d_{1,1} & d_{1,2} & \cdots & d_{1,N} \end{pmatrix} \end{aligned} \tag{8.5}$$

where $Q \in \mathbb{R}^{12\times M}$ and $D \in \mathbb{R}^{12\times N}$. Now we can compute the matrix

$$C = \frac{1}{M} \cdot (1^{M\times N} - Q^T \cdot D) \tag{8.6}$$

where $1^{M\times N} \in \mathbb{R}^{M\times N}$ is the matrix for which all entries are 1. The matrix C has the form

$$\begin{aligned} C &= \tfrac{1}{M} \cdot \begin{pmatrix} 1 - \langle \vec{q}^M, \vec{d}^1 \rangle & & & 1 - \langle \vec{q}^M, \vec{d}^N \rangle \\ \vdots & & \iddots & \\ 1 - \langle \vec{q}^2, \vec{d}^1 \rangle & 1 - \langle \vec{q}^2, \vec{d}^2 \rangle & & \\ 1 - \langle \vec{q}^1, \vec{d}^1 \rangle & 1 - \langle \vec{q}^1, \vec{d}^2 \rangle & \cdots & 1 - \langle \vec{q}^1, \vec{d}^N \rangle \end{pmatrix} \\ &= \tfrac{1}{M} \begin{pmatrix} \delta(\vec{q}^M, \vec{d}^1) & & & \delta(\vec{q}^M, \vec{d}^N) \\ \vdots & & \iddots & \\ \delta(\vec{q}^2, \vec{d}^1) & \delta(\vec{q}^2, \vec{d}^2) & & \\ \delta(\vec{q}^1, \vec{d}^1) & \delta(\vec{q}^1, \vec{d}^2) & \cdots & \delta(\vec{q}^1, \vec{d}^N) \end{pmatrix} \end{aligned} \tag{8.7}$$

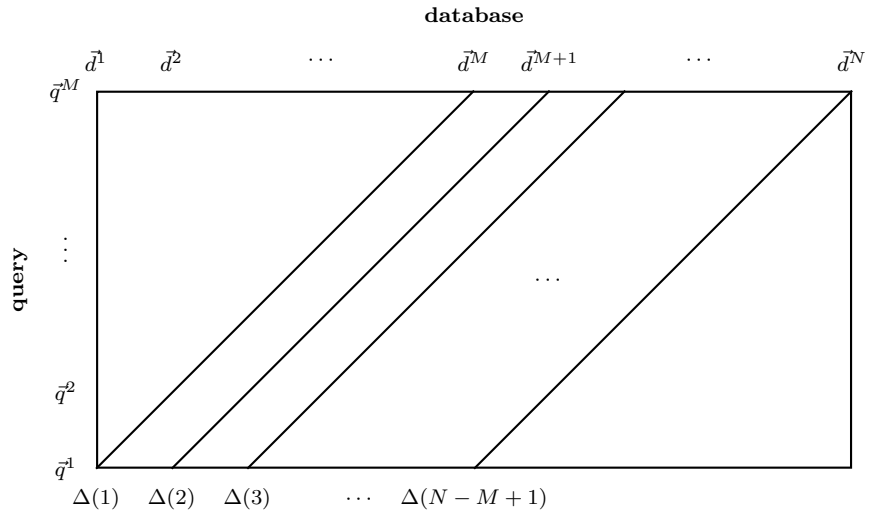

Figure 8.2. An abstract representation of the matrix C. The diagonal lines represent the entries that are summed up to get the entries of Δ.

Therefore we have that

$$\begin{aligned} \sum_{k=1}^{M} C_{k,i+k-1} &= \tfrac{1}{M} \cdot \sum_{k=1}^{M} \delta(\vec{q}^k, \vec{d}^{i+k-1}) && (8.7) \\ &= \delta^M((\vec{q}^1, \vec{q}^1, ...\vec{q}^M), (\vec{d}^i, \vec{d}^{i+1}, ...\vec{d}^{i+M-1})) && (8.2) \\ &= \Delta(i) && (8.3) \end{aligned} \tag{8.8}$$

This computation is visualized in Figure 8.2. To find the subsequence of chroma vectors in the database that is maximal similar to the query sequence of chroma vectors we can now simply pick the index i in Δ such that $\Delta(i)$ is minimal. The index i then marks the first vector of the best matching subsequence in the database. Figure 8.3 shows an example of the matching process.

8.3 Query Pool

To increase the possibility of finding a match in the database that is very similar in its harmonic progression to the transition region it is helpful to give the matching process some degrees of freedom. These degrees of freedom are musically motivated and can therefore be expressed as modifications of the beat-synchronous chroma features of the transition region. We separate the degrees of freedom into three subclasses.

The *position tolerance* allows to temporally shift the transition region by up to a certain amount of beats. This relaxes the constraint on where exactly the transition to another audio recording should take place (see Figure 8.4).

The *pitch tolerance* accounts for the fact that it is possible to pitch shift every audio recording in the database by a small amount of semitones without significant audio degra-

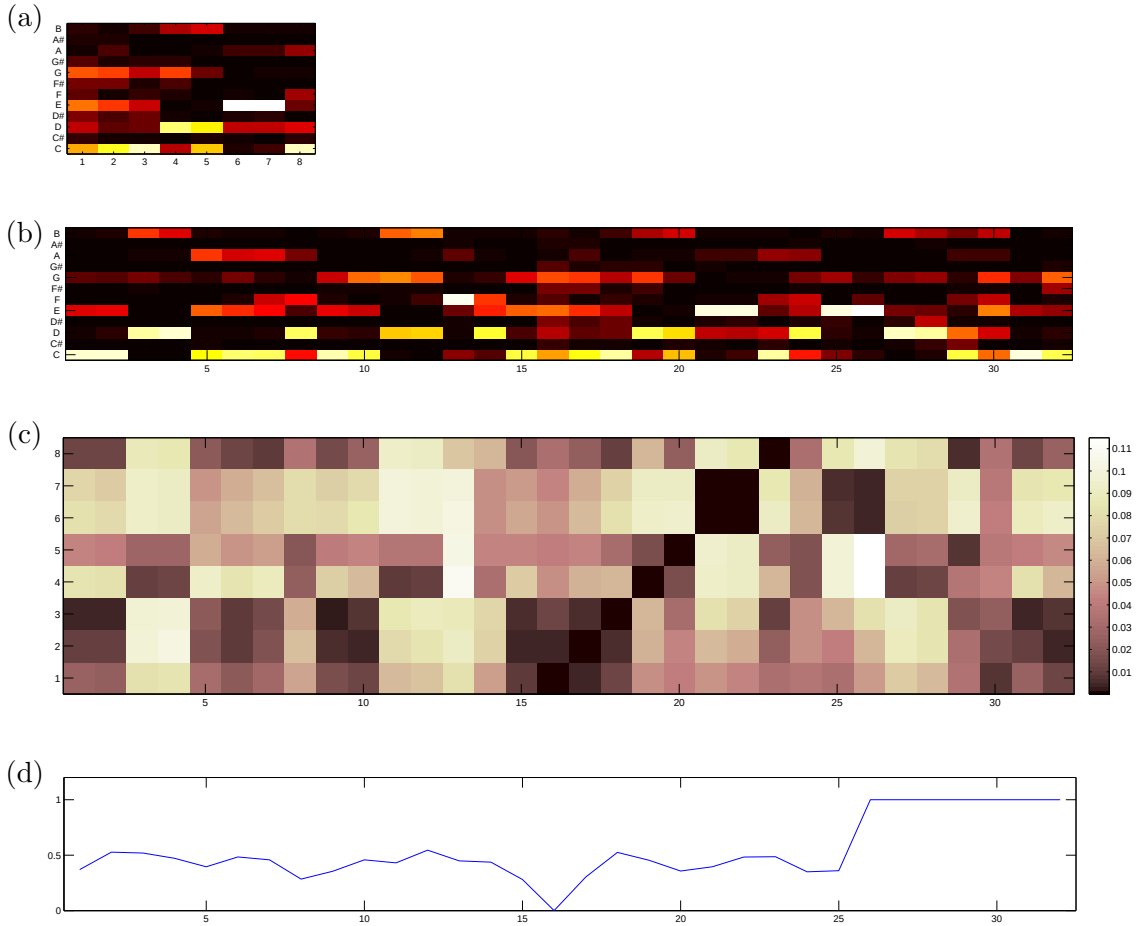

Figure 8.3. **(a)** The query sequence of chroma vectors Q computed from 8 measures of the song "let it be" from the Beatles. **(b)** The database sequence of chroma vectors D computed from the first 32 measures of "let it be" from the Beatles. Again the database is extremely short in this example for the purpose of simplicity. **(c)** The matrix C computed from Q and D. One can already see the low-distance-diagonal starting at database index 16. **(d)** The distance function Δ computed from C. On can observe that at index 16 the value of the function is 0. Therefore the subsequence in D starting at index 16 is perfectly identical to the query. In fact the query sequence of chroma vectors was a subsequence of the database sequence of chroma vectors in the first place.

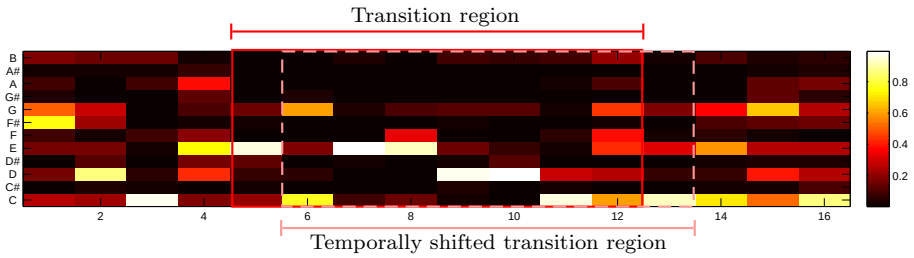

Figure 8.4. Application of the position tolerance. The position of the first vector in the query sequence of chroma vectors is shifted to the right by one beat.

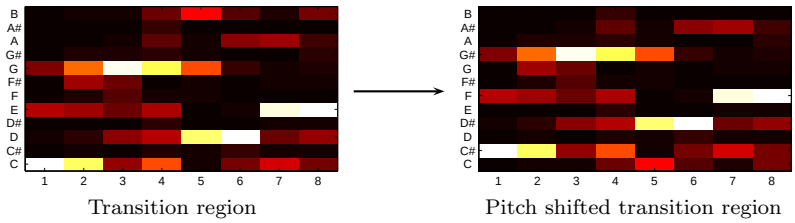

Figure 8.5. Application of the pitch tolerance. By vertically cyclic shifting the chroma vectors by one, a match for this modified sequence of chroma vectors that is found in the database is harmonically similar to the transition region if pitched down by one semitone.

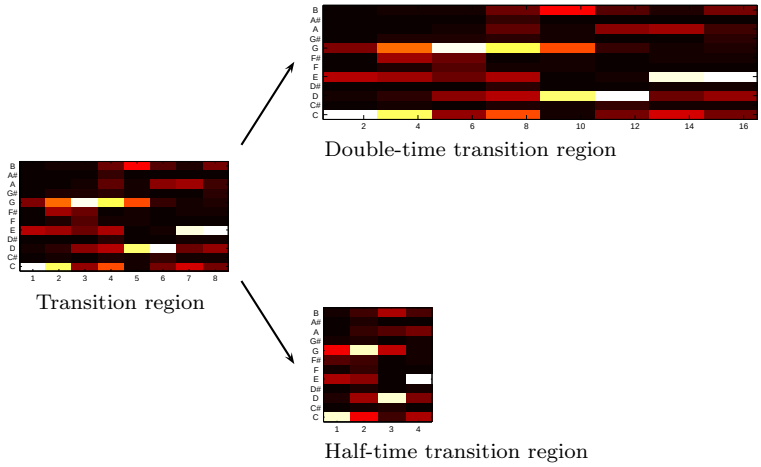

Figure 8.6. Application of the type of meter tolerance. By either expanding the sequence of chroma vectors by doubling every chroma vector or compressing it by averaging two neighbored chroma vectors it is possible to find harmonically similar matches in the database that have a different beat resolution than the original transition region.

dation. To emulate such a pitch shift, chroma vectors of the transition region are simply cyclic shifted vertically by the desired amount of semitones (see Figure 8.5).

The *type of meter tolerance* allows for changing the beat resolution during the transition phase to half-time or double-time. This is helpful since the the beat of an audio recording may be perceived on different resolutions and is therefore ambiguous. Without the type of meter tolerance, two recordings that share the exact same harmonic progression can not be matched in case the beat of the two audio recordings is annotated on different resolutions. To emulate the different beat resolutions, the sequence of chroma vectors of the transition region is either expanded by doubling every single chroma vector (double-time) or compressed by averaging two neighbored chroma vectors (half-time) (see Figure 8.6).

By applying these modifications and combinations of them to the original query sequence of chroma vectors we extend the single query sequence to a *query pool*. Instead of matching only one sequence of chroma vectors with the database, we sequentially match all sequences in the query pool with the database and pick the one that yielded the match with the

lowest distance value in the end.

8.4 Database

The matching technique described in Section 8.2 needs the database to have the form of a single long sequence of chroma vectors. Therefore the chromagrams of all audio recordings in the database are concatenated and we keep track of the boundaries in a separate data structure. Additionally, we store further informations about every included audio recording like for example the name of the sound file and the temporal positions of all beats on which the computation of the beat synchronous chroma features was based. The latter information is very important since the beat synchronous chroma features do not hold any temporal information anymore by themselves.

A problem that might occur when representing the database as a concatenation of the chromagrams of all includes audio recordings is that the subsequence of chroma vectors in the database that is most similar to the query sequence of chroma vectors might contain chroma vectors from two different audio recordings. The solution to this problem is to pad the sequences of chroma vectors of all audio recordings with a sequence of zero-vectors which is at least of the same length as the query sequence of chroma vectors. The effect of this zero-pad is visualized in Figure 8.7.

8.5 Warping

Having developed a good TSM algorithm in Chapter 6, we now can use this algorithm to rhythmically synchronize the transition region and the matched region. This is done by a temporal alignment of the beat positions of the two audio clips. The goal is therefore to compute two time-stretch functions τ_{tr} for the transition region and τ_{mr} for the matched region such that when overlaying the results of the time-scale modifications of both audio clips their beat positions are aligned. Figure 8.8 shows an example. Note that both the transition region as well as the matched region have the same length on the beat level by design of the matching process. We define this number of beats to be N. Let the sequence of beat positions in the transitions region given in seconds be

$$T = t_1, t_2, ..., t_N \tag{8.9}$$

and the sequence of beat positions in the matched region

$$M = m_1, m_2, ..., m_N \tag{8.10}$$

while it holds that $q_1 = 0$ and $m_1 = 0$. We now have to define a sequence of beat positions

$$S = s_1, s_2, ..., s_N \tag{8.11}$$

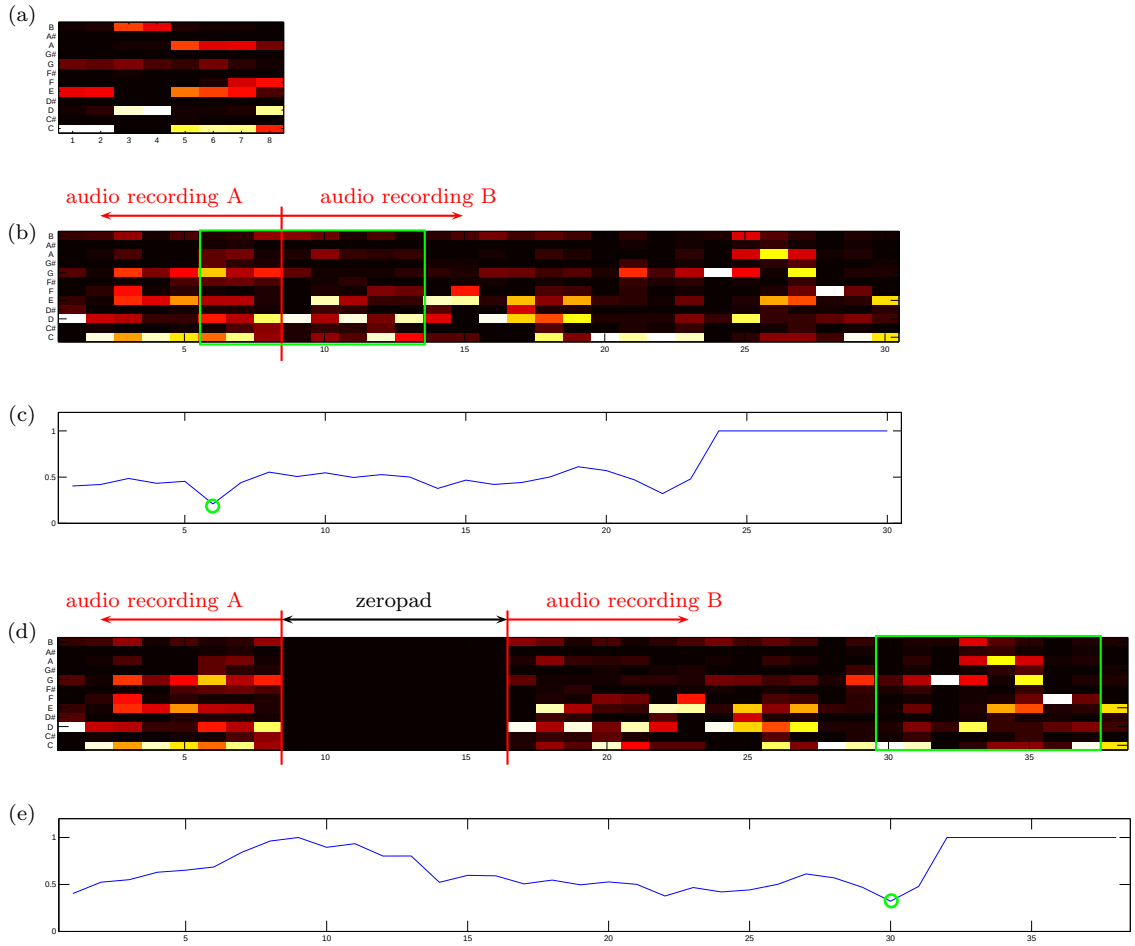

Figure 8.7. **(a)** The sequence of query chroma vectors. **(b)** A sequence of database chroma vectors consisting of two audio recordings. The chromagrams are not separated by a zero pad. **(c)**. The distance function Δ computed from the query chromagram from (a) and the database chromagram from (b). The subsequence of chroma vectors in the database that is most similar to the query sequence of chroma vectors (marked by green box in (b)) contains chroma vectors from two different audio recordings. **(d)** The database chromagram from (b) with a zeropad between the two songs. **(e)** The distance function Δ computed from the query chromagram from (a) and the database chromagram from (d). The zeropad prevents the distance function from having small values at audio recording boundaries.

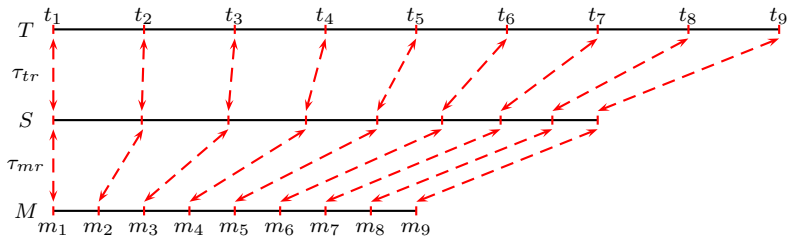

Figure 8.8. An example of the computations of τ_{tr} and τ_{mr}.

that we can use to define the time-stretch functions τ_Q and τ_M by sets of anchor points

$$\begin{aligned} \tau_Q &= ((t_1, s_1), (t_2, s_2), ..., (t_N, s_N)) \\ \tau_M &= ((m_1, s_1), (m_2, s_2), ..., (m_N, s_N)) \end{aligned} \tag{8.12}$$

The sequence S should be chosen such that the tempo of the transition region smoothly adapts to the tempo of the matched region. We therefore define

$$\begin{aligned} &s_1 = 0 \\ &s_i = s_{i-1} + \tfrac{N-(i-1)}{N} \cdot (t_i - t_{i-1}) + \tfrac{i-1}{N} \cdot (m_i - m_{i-1}) \quad \text{for } i \in [2 : N] \end{aligned} \tag{8.13}$$

The transition region as well as the matched region are then time-scale modified using the time-stretch functions τ_{tr} and τ_{tm} respectively.

8.6 Blending

To finish the transition we now overlay the transient region with the matched region and apply a simple blending. Blending is the process of making the transition between two audio recordings less abrupt. The simplest kind of blending is the so called *crossfade*. By decreasing the volume of the transition region until it is finally not perceivable any more while at the same time increase the volume of the matched region from silence to the normal volume, a smooth transitions between the two clips is created. Although there also exist more advanced blending techniques the simple crossfade technique suffices our needs.

The whole pipeline of the CONNECTOR is summarized in Figure 8.6.

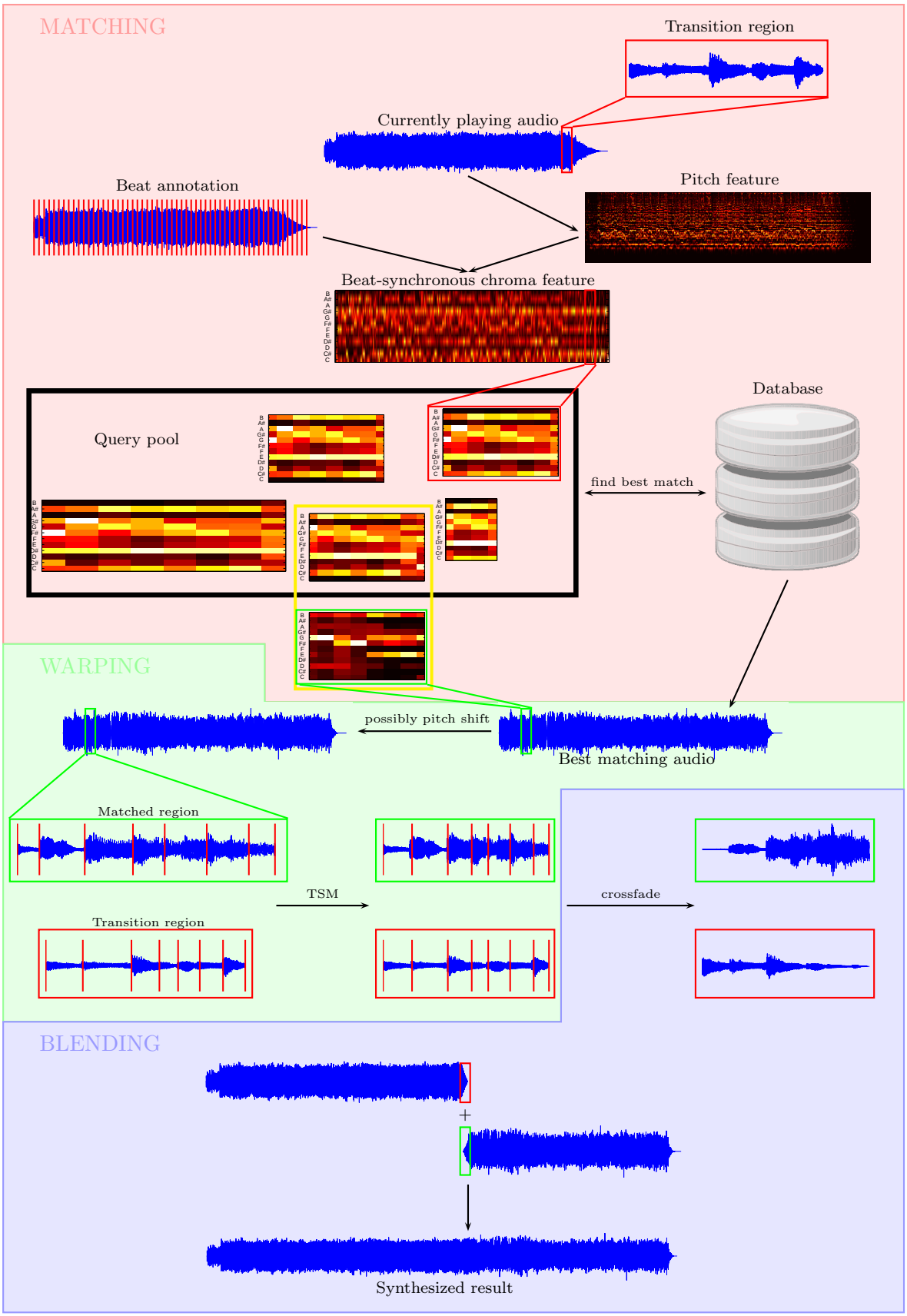

Figure 8.9. Overview of the CONNECTOR.

Chapter 9

Future Work

In this thesis, we discussed various time-scale modification algorithms and their application to music audio signals. In particular, we analyzed WSOLA as an example of a time-domain algorithm and the Phase Vocoder as an example of a frequency domain algorithm. Furthermore we introduced modifications to the WSOLA algorithm which reduce artifacts in the algorithm's output by preserving transients during the time-scale modification process. Although this already increases the quality of WSOLA significantly, we feel that there is still a lot of room for improvements. As already briefly discussed in Chapter 4, WSOLA often has problems when time-scale modifying signals of high complexity like for example recordings of orchestral music. This kind of audio signals are highly polyphonic due to the presence of many instruments that play different notes. Therefore numerous fundamental frequencies exist in such signals. However, WSOLA is only capable of preserving a single fundamental frequency. As a consequence, the remaining fundamental frequencies are not preserved properly and phase jump artifacts are introduced which manifest themselves in a metallic, noise-like sound. An idea to overcome this phenomenon is to split the input signal into several subbands prior to the time-scale modification. The subbands are then time-scale modified seperatly and added up afterwards. Every single subband is of course less complex than the original input signal. Therefore the time-scale modification of a single subband should yield significantly less artifacts than the time-scale modification of the original signal and therefore also the sum of all time-scale modified subband signals should suffer less from artifacts. This idea is inspired by the Phase Vocoder which ensures phase continuity for a very fine-grained frequency decomposition of the input signal. But this fine decomposition is also responsible for the loss of vertical phase coherence in the output signal which causes the artifacts that are typical for the Phase Vocoder. By choosing the number of subbands to be rather low, namely to be in the magnitude of the number of fundamental frequencies in the input signal, the loss of vertical phase coherence should be prevented.

On the application side we see a lot of potential in the CONNECTOR. For example, although our matching technique already yields matches that are of a similar harmonic content as the transition region, it could be further improved by techniques from the field of *chord recognition*. During the matching process, we select matches based on the assumption that the similarity of two sequences of chroma vectors yields a strong indication for harmonic

similarity. But in fact the numerical similarity of the sequences of chroma vectors is no guarantee for a perceptual similarity. For example the major and minor chords for a certain root note only differ in a single chroma value and may therefore be matched. But changing the harmony from major to minor during a transition often sounds unpleasant. Furthermore it is likely that the euphony of a transition is not only dependent on the harmonic structure of the transition region, but also on the chord progression on a larger scale.

A further improvement of the CONNECTOR can be done in its blending stage. The simple crossfade that we apply during the transition can be replaced by some more advanced blending technique. Experiments have shown that it is for example sometimes convenient to blend different instruments in an audio signal in a different way or at different points in time. To this end, *source separation* techniques can be applied to the audio signals to get the functionality of an *instrument equalizer*.

In conclusion, we have seen that time-scale modification is a central topic in audio signal processing with various applications such as automated soundtrack generation. In particular, in the context of the CONNECTOR, it has shown that time-scale modification needs to be combined with further music analysis and retrieval tasks, including beat tracking, chord recognition, source separation and audio matching. For the future, we plan to further automate the various components of the CONNECTOR and to develop tools that allow users to easily search for, modify, merge, and generate suitable soundtracks.

Appendix A

Source Code

In this chapter, the header of our MATLAB implementation of TP-WSOLA that was described in Chapter 6 and that was created during the writing of this thesis is given.

Sample usage:

```
parameter.automatedTransientPreservation = 1;
y = tp_wsola(x, 2, parameter);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Name: tp_wsola
% Version: 1.0
% Date: 5.10.2011
% Programmer: J. Driedger (driedger@mpi-inf.mpg.de)
%
% Description:
%   tp_wsola is a transient preserving time-scale modification algorithm
%   based on the WSOLA algorithm. It is capable of preserving transients
%   automatically as well as preserving transients at additionally given
%   positions.
%
% Input:
% y = wsola(x,t):
%   x:  mono / stereo signal (column vector)
%   t:  either a (n x 2) matrix containing anchor points specified in
%       samples or seconds or a constant time-stretch factor.
%
% Optional input arguments:
% y = wsola_core(x,t,parameter):
%   parameter.windowSize: specifies the size of the Hann-window that is
%                         used for input and output.
%   parameter.tolerance : specifies the tolerance Delta_max for the input
%                         window positions.
%   parameter.APStyle   : Is either 'samples' or 'seconds' and specifies
%                         how the given anchor points are interpreted.
%   parameter.samplerate: specifies the samplingrate of the audio.
%   parameter.automatedTransientPreservation:
%                         Should be set to 1 in case transients should be
```

```
%                           detected and preserved automatically.
%   parameter.preservedPositions:
%                           Additional positions of transients that should be
%                           preserved.
%   parameter.transientWindowSize:
%                           The size of the transient window Xi.
%
% Output:
%   y: time-scale midified output signal (mono / stereo)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Appendix B

Listening Test Questionnaire

The questionnaire that was used for the listening test to rate the quality of different time-scale modification algorithms (Section 7).

Instructions

The aim of this evaluation is to rate the quality of different audio time-scale modification techniques. Time-scale modification algorithms allow us to change the speed on which an audio recording is played without altering the pitch of the audio. They are intended to produce versions of an audio recording that sound as if the musical piece was initially performed on a different tempo. In the following we will present you with a couple of different audio recordings along with several time-scale modified versions of them which were produced using different time-scale modification algorithms. The original audio recordings were created synthetically and may therefore sound a little unnatural. All modified versions are double the length of the original audio recordings. We ask you to rate the quality of the manipulated versions on a scale from 5 (excellent) to 1 (bad). To give you a reference for your evaluation there will always be a version that is simply the original recording synthesized on half the tempo. This version can therefore be seen as the result of a *perfect* time-scale modification algorithm.
You will be able to listen to the original audio recording and the modified versions as often as you like but it is important that you listen to each audio recording and each modification at least once from the beginning to the end. Feel also free to comment on everything that occurs to you during the listening test.

EXAMPLE

	excellent 5	good 4	fair 3	poor 2	bad 1	Comment
Version 1	○	⊗	○	○	○	something was wrong with the pitch
Version 2	○	⊗	○	○	○	
Version 3	⊗	○	○	○	○	sounded perfect!
Version 4	○	○	○	○	⊗	had nothing to do with the original...
Version 5	○	○	○	⊗	○	Beginning was good, but screwd up at the end

Additional Comments: The quality was in general hard to tell since this example is pretty stupid.

Appendix C

Listening Test Comments

In the following the comments that were given by the listening test participants during the listening test are listed.

Name	Algorithm	Comments
BASS1	MPEX4	slightly wrong notes — slightly stuttering at the beginning but 2nd half was good — strange chorus like effect — quick tone changes do not sound well — quite an echo these artifacts
	trans. prev. WSOLA	artifacts at the beginning/end — slightly wrong notes — slightly stuttering at the beginning but 2nd half was good — at the beginning of each tone quite some artifacts
	WSOLA	artifacts in higher pitches — unclean note transitions at the end — slightly stuttering — a little bit vibrating — artifacts — at the beginning of each tone there were artifacts
	Phase Vocoder	lower volume? — no clean tones — stuttering — dynamics killed? — higher frequencies cut off — because the sound is vibrating — attacks do not sound well — somehow crumbly
BONGO1	MPEX4	delay effect, small artifacts on higher drums — echo/doubled notes — slight stuttering — strange onset. sounds like after using a compressor — attacks are duplicated — the sound is interesting but not right
	trans. prev. WSOLA	higher pitches bad, lower good — slight echo — the beats sound stretched — stretches too long (bongos sound artificial) — the sound lost some corpus — it sounds empty
	WSOLA	electric sound — stuttering — metallic sound — seems to have noise — attacks are duplicated — doesnt sound like a bongo anymore
	Phase Vocoder	echo — doubled notes — one additional hit every time like short delay — pingpong echo — echo of sound happens — attacks are duplicated — too much echo
DRUM1	MPEX4	still lots of artifacts but sounds more homogeneous than others — stuttering — metallic + wobble — wobbly artifacts — electro drums, but bad bass drum
	trans. prev. WSOLA	sounds like a synthesizer at higher drums — vibrating notes — stuttering disturbing sound — sounds like beat-mashed — the drum noises sound as if they are duplicated — the drum became a distorted mess
	WSOLA	but sounds funny — metallic — stuttering, disturbing sound — metallic — sound quality too low, too much noise — the drum noises sound as if they are duplicated — electro!!!
	Phase Vocoder	sounds like a wet room — sligtly metallic — echoing — echo/delay — a little bit echo but okay — the drum noises sound as if they are duplicated — echo

FLUTE1	MPEX4	onsets sounds a little like a glockenspiel — difference to original hardly noticableonly at the end — vibrating at the ending note — almost perfect but a bit too long
	trans. prev. WSOLA	little artifact at the end — difference to original hardly noticable — vibrating at the ending note
	WSOLA	little artifact at the end + glockenspiel onsets — difference to original hardly noticable only in the beginning — vibrating at the ending note —could be the original but the long tone at the end sounds strange
	Phase Vocoder	artifacts at onsets — doubled onset at each note — chorus — vibrating at the ending note — something is strange
GENRE1	MPEX4	drums bad, rest good — metallic drums — stuttering but better than others — wobbling — quite good but the instruments do not seem synchronized
	trans. prev. WSOLA	good drums at the end replaced by electric guitar — stuttering — duplicated attacks — almost perfect but the guitar at the end is strange
	WSOLA	lot of artifacts in snare — metallic drums — stuttering — metallic — duplicated attacks — echo and distortion
	Phase Vocoder	sounds like wet room — metallic and dull — low volume echo — not much survives here — bass guitar almost gone at the end — unbearable, too many artifacts — completely broken
JAZZ1	MPEX4	still some artifacts — sounds “blurry” — ringing sound pitch? — although it seems strange — some reverb
	trans. prev. WSOLA	Notes somehow wrong, anoying peeptone — distorted — pitch unstable — weird noise after some of the played notes — noise in the pitch — ”plastic” sound — too distorted
	WSOLA	lots of artifacts from onsets — notes somehow wrong —distorted — weird noise after some of the played notes — too much noise in the pitch — duplicated attacks + “plastic” — almost broken, cannot stand the chords
	Phase Vocoder	wet room honkey tonk effect — echo, metallic — like a distant fog piano but almost “good” — postnuclear piano — nearly cannot tell the instrument — sounds like “far away” — sounds like plastic
OWN1	MPEX4	wet room — tretching notes misses the fundamental frequency somehow — stuttering but slightly better — less echo(?) — but there are slight artifacts
	trans. prev. WSOLA	very good you hear that it is the streched version mainly because of the first tone (Einschwingphase) and the longer echo — small shortcomings at the end with the note that gets louder — some new sound at offbeatspitch bends at the end — like 1 with some reverb — there is some artifact, but it is positive (it has a positive effect)
	WSOLA	lots of artifacts in onsets — stretching notes misses the fundamental frequency somehow — stuttering — echo — sound quality is low, also the noise happens more in the end — sounds funny not authentic but cool
	Phase Vocoder	echo lower volume — echo sounds weird — much echo! — sounds very dull — nearly cannot capture the tone hight — empty and echo
POP3	MPEX4	very beginning sounded good and in between — minimal stuttering — piano got chopped — the noise mixed in pitch are quite annoying — sounds a bit “hollow”
	trans. prev. WSOLA	good in the middle — metallic, trembling notes — stuttering — piano got chopped — keyboard sounds wierd — the noise mixed in pitch are quite annoying — duplicated attacks — where did the instruments go?
	WSOLA	metallic trembling notes — distortion — piano got chopped — the noise mixed in pitch are quite annoying — duplicated attacks, especially bad with piano — completely broken
	Phase Vocoder	wet room/lower volume — echo , metallic different — too many artifacts — empty and falling apart
POP4	MPEX4	slightly metallic — drums bad, rest good — cool reverb — poor vibrato

	trans. prev. WSOLA	distorted guitars — drums worse at end — drums in the end sound wrong — distorted
	WSOLA	onset artifacts — distorted guitar — guitar doesnt get hurt too much — drums at the end noisy — beginning is fine. at the end the drum sound is pretty noisy — the vibrato visits too many other scores
	Phase Vocoder	wet room — echowrong notes — destroyed — sounds dull — sound quality is to low — not clear — broken
VIOLIN1	MPEX4	half tempo vibrato is horrible — vibrato is painfully slow — slightly metallic — pitch is not clear — pitch? — the vibrato is broken
	trans. prev. WSOLA	sounds like onsets artifacts all over the piece — metallic half tempo vibrato is horrible — distortion — slightly metallic — pitch is not clear — metallic sound — the vibrato is broken and even distorted
	WSOLA	sounds like onsets artifacts all over the piece but stronger — metallic half tempo vibrato is horrible — distortion — slightly metallic — plastic — poor vibrato
	Phase Vocoder	wet room/alters sound — echometallic half tempo vibrato is horrible — like chorus effect — flanger effect — far away — this one is just empty and plastic like

Appendix D

The Connector

We implemented the Connector like described in Chapter 8 as a plugin for the MATLAB audio player developed by Philipp von Styp-Rekowsky [44]. Our implementation offers a graphical user interface that allows for defining all necessary inputs to our tool like the position and length of the transition region, the several degrees of freedom in the matching process or the utilized database. Figure D.1 shows a screenshot of the tool.

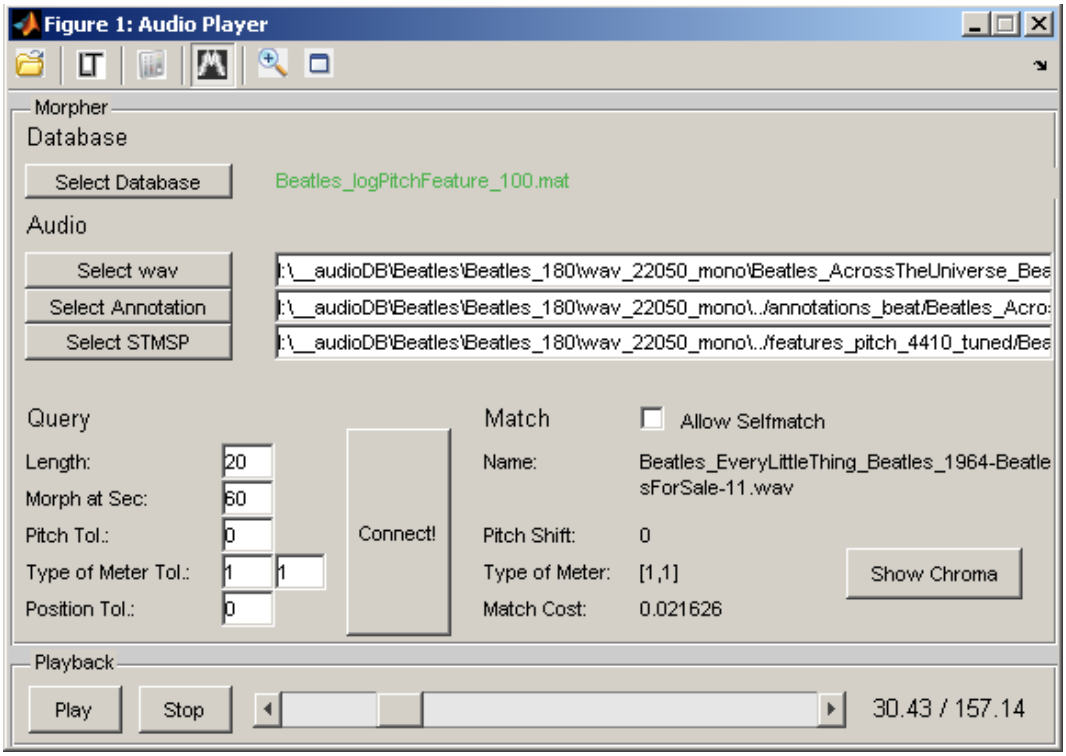

Figure D.1. Screenshot of the Connector.

Bibliography

[1] D. Barry, D. Dorran, and E. Coyle, *Time and pitch scale modification: A real-time framework and tutorial*, in Proceedings of the 11th International Conference on Digital Audio Effects (DAFx-08), 9 2008.

[2] 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, 13 (2005), pp. 1035–1047.

[3] D. Cliff, *Hang the dj: Automatic sequencing and seamless mixing of dance-music tracks*, tech. report, HP Laboratories Bristol, 2000.

[4] N. Collins, *A comparison of sound onset detection algorithms with emphasis on psychoacoustically motivated detection functions*, in AES Convention 118, Barcelona, Spain, 2005.

[5] ——, *Using a pitch detector for onset detection*, in Proceedings of the International Conference on Music Information Retrieval (ISMIR), London, UK, 2005, pp. 100–106.

[6] D. Cope, *Experiments in Musical Intelligence*, A-R Editions, Inc., 1996.

[7] N. Degara, A. Pena, M. E. P. Davies, and M. D. Plumbley, *Note onset detection using rhythmic structure*, in Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Dallas, TX, USA, 2010, pp. 5526–5529.

[8] S. Dixon, *Evaluation of the audio beat tracking system beatroot*, Journal of New Music Research, 36 (2007), pp. 39–50.

[9] M. Dolson, *The phase vocoder: A tutorial*, Computer Music Journal, 10 (1986), pp. 14–27.

[10] M. Dolson and J. Laroche, *Improved phase vocoder time-scale modification of audio*, IEEE Transactions on Speech and Audio Processing, 7 (1999), pp. 323–332.

[11] D. Dorran, E. Coyle, and R. Lawlor, *Audio time-scale modification using a hybrid time-frequency domain approach*, in Proceedings Workshop on Applications of Signal Processing (WASPAA), New Paltz, New York, USA, oct 2005.

[12] D. P. W. Ellis, *A phase vocoder in Matlab*, 2002. Web resource, last consulted in October 2011.

[13] ——, *Beat tracking by dynamic programming*, Journal of New Music Research, 36 (2007), pp. 51–60.

[14] J. L. Flanagan and R. M. Golden, *Phase vocoder*, Bell System Technical Journal, 45 (1966), pp. 1493–1509.

[15] H. Friedman, *Variable speech*, Creative Computing, 9 (1983), p. 122.

[16] L. Gareth, *Musimathics: The Mathematical Foundations of Music, Volume 1*, The MIT Press, 2006.

[17] M. Goto, H. Hashiguchi, T. Nishimura, and R. Oka, *RWC music database: Popular, classical and jazz music databases*, in Proceedings of the International Conference on Music Information Retrieval (ISMIR), Paris, France, 2002.

[18] P. Gournay, R. Lefebvre, and P.-A. Savard, *Hybrid time-scale modification of audio*, in Audio Engineering Society Convention 122, 5 2007.

[19] S. Grofit and Y. Lavner, *Time-scale modification of audio signals using enhanced wsola with management of transients*, IEEE Transactions on Audio, Speech & Language Processing, 16 (2008), pp. 106–115.

[20] P. Grosche and M. Müller, *Extracting predominant local pulse information from music recordings*, IEEE Transactions on Audio, Speech and Language Processing, 19 (2011), pp. 1688–1701.

[21] P. Grosche, M. Müller, and C. S. Sapp, *What makes beat tracking difficult? A case study on Chopin Mazurkas*, in Proceedings of the 11th International Conference on Music Information Retrieval (ISMIR), Utrecht, Netherlands, 2010, pp. 649–654.

[22] G. Haupenthal, *Geschichte der Würfelmusik in Beispielen*, PhD thesis, Universität des Saarlandes, 1994.

[23] R. Hudson, *Stolen time: the history of tempo rubato*, Clarendon paperbacks, Clarendon Press, 1994.

[24] T. Jehan, *Creating Music by Listening*, PhD thesis, Massachusetts Institute of Technology, 2005.

[25] N. Juillerat, S. M. Arisona, and S. Schubiger-Banz, *A hybrid time and frequency domain audio pitch shifting algorithm*, in Audio Engineering Society Convention 125, 10 2008.

[26] G. Kaiser, *A Friendly Guide to Wavelets*, Birkhäuser, Boston, 1994.

[27] A. Klapuri, *Sound onset detection by applying psychoacoustic knowledge*, in Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Washington, DC, USA, 1999, pp. 3089–3092.

[28] A. P. Klapuri, A. J. Eronen, and J. Astola, *Analysis of the meter of acoustic musical signals*, IEEE Transactions on Audio, Speech and Language Processing, 14 (2006), pp. 342–355.

[29] P. Leveau, L. Daudet, and G. Richard, *Methodology and tools for the evaluation of automatic onset detection algorithms in music*, in Proceedings of the International Conference on Music Information Retrieval (ISMIR), Barcelona, Spain, 2004.

[30] H.-L. Lin, Y.-T. Lin, M.-C. Tien, and J.-L. Wu, *Music paste: Concatenating music clips based on chroma and rythm features*, in 10th International Society for Music Information Retreival Conference (ISMIR 2009), 2009, pp. 213–218.

[31] A. Moinet and T. Dutoit, *PVSOLA: a phase vocoder with synchronized overlap-add*, in Proceedings of the 14th International Conference on Digital Audio Effects (DAFx-11), Paris, France, 2011, pp. 269–275.

[32] E. Moulines and F. Charpentier, *Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones*, Speech Communication, 9 (1990), pp. 453 – 467.

[33] M. Müller, *Information Retrieval for Music and Motion*, Springer Verlag, 2007.

[34] M. Müller, V. Konz, N. Jiang, and Z. Zuo, *A multi-perspective user interface for music signal analysis*, in Proceedings of the International Computer Music Conference (ICMC), Huddersfield, England, UK, 2011.

[35] M. Müller, F. Kurth, and M. Clausen, *Audio matching via chroma-based statistical features*, in Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR), 2005, pp. 288–295.

[36] T. H. Park, *Introduction to Digital Signal Processing: Computer Musically Speaking*, World Scientific, 2010.

[37] S. Roucos and A. Wilgus, *High quality time-scale modification for speech*, in Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Tampa, Florida, USA, 1985, pp. 236–239.

[38] D. Schwarz, *A system for data-driven concatenative sound synthesis*, in Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-00), Verona, Italy, july 2000.

[39] ——, *The caterpillar system for data-driven concatenative sound synthesis*, in Proceedings of the 6th International Conference on Digital Audio Effects (DAFX-03), London, UK, september 2003.

[40] ——, *Current research in concatenative sound synthesis*, in Proceedings of the International Computer Music Conference (ICMC), Barcelona, Spain, september 2005.

[41] ——, *Concatenative sound synthesis: The early years*, Journal of New Music Reaserch, 35 (2006).

[42] W. A. Sethares, *Rhythm and Transforms*, Springer Publishing Company, Incorporated, 1st ed., 2007.

[43] W. Verhelst and M. Roelands, *An overlap-add technique based on waveform similarity (wsola) for high quality time-scale modification of speech*, in Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Minneapolis, USA, 1993.

[44] P. von Styp-Rekowsky, *Towards time-adaptive feature design in music signal processing*, master's thesis, Saarland University, 2011.

[45] S. Wenger and M. Magnor, *Constrained example-based audio synthesis*, in Proceedings of the 2011 IEEE International Conference on Multimedia and Expo (ICME 2011), Barcelona, Spain, July 2011.