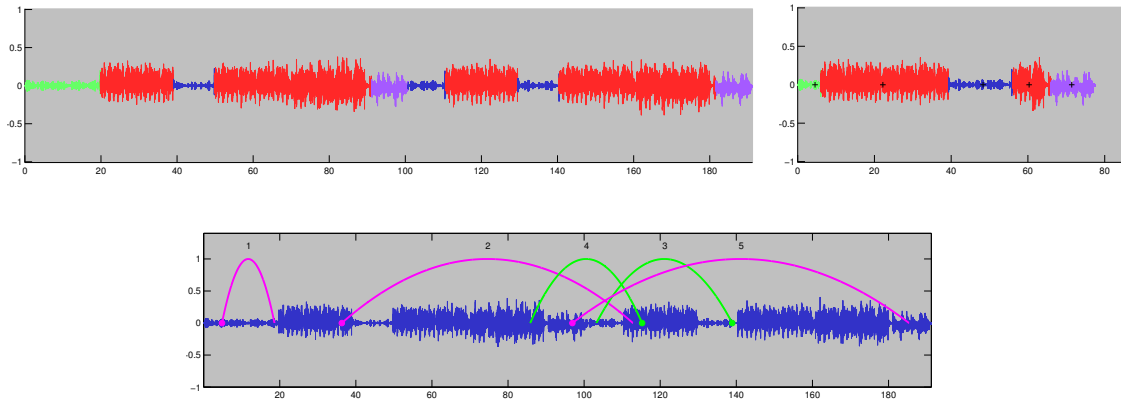
Music Retargeting and Synthesis



Simon Wenner

Master Thesis
April 2012

Supervisors:
Dr. Jean-Charles Bazin
Dr. Alexander Hornung

Advisor:
Prof. Dr. Markus Gross

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

cgl
Computer Graphics Laboratory ETH Zurich

Abstract

Music is an important part of many computer graphics applications like animation, film or computer games. Yet, many advancements in computer graphics have not been generalized appropriately to the audio domain. In this thesis we propose a novel method for dynamic content-aware rescaling and reassembly of music. The method is inspired by the recent works in image retargeting, video reshuffling and character animation. Given a certain target length and additional constraints, such as music structure or importance, the proposed method performs a global optimization to find an assembly of the input music that fulfills the constraints with the best possible quality. Low and high-level features are extracted from the audio signal in order to build a graph, which encodes the constraints and potential forward and backward jumps. The desired music is synthesized using a k-stops shortest path algorithm. The algorithm runs in real time and it creates results with imperceptible transitions. We show applications such as shortening and enlarging the duration (up to infinity), music reshuffling, removal of vocal parts and video editing with audio alignment.

Zusammenfassung

Musik ist ein wichtiger Bestandteil von Anwendungen der Computergrafik wie Animation, Film oder Videospiele. Trotzdem wurden viele der Erkenntnisse aus der Computergrafik nicht genügend generalisiert um sie auf Audiodaten anwenden zu können. In dieser Masterarbeit schlagen wir eine neuartige Methode vor, um Musik dynamisch, inhaltsabhängig zu skalieren und umzustrukturieren. Die Methode ist inspiriert von aktuellen Forschungen im Bereich der inhaltsabhängigen Bildverzerrung, der Umstrukturierung von Videos und der Charakteranimation. Der Benutzer definiert eine gewünschte Dauer und zusätzliche Bedingungen, wie zum Beispiel eine Musikstruktur oder eine Gewichtung, so führt die vorgeschlagene Methode eine globale Optimierung durch, um eine Anordnung der Eingabemusik zu finden, welche alle Bedingungen mit bestmöglicher Qualität erfüllt. Charakteristiken verschiedener Granularitäten werden aus dem Audiosignal extrahiert um einen Graphen aufzubauen, der die Beschränkungen und die potenziellen vorwärts und rückwärs Sprünge beschreibt. Die gewünschte Musik wird mittels eines K-Stops-kürzesten-Pfadalgorithmus synthetisiert. Der Algorithmus funktioniert in Echtzeit und er erzeugt Resultate mit nicht wahrnehmbaren Übergängen. Wir präsentieren eine Vielzahl von Anwendungen, wie das Verkürzen und das Verlängern der Abspieldauer (bis zu unendlich), die Musikumstrukturierung, die Entfernung von Teilen mit Gesang und die Videobearbeitung mit Audiosynchronisation.

Acknowledgments

First of all I would like to thank Jean-Charles Bazin and Alexander Hornung for supervising my thesis. The frequent discussions where always very interesting and a great source of ideas. I thank Changil Kim for his help with the dynamic programming algorithm, Reto Troxler[1] for the custom video material of the song "Better World" (released 2010) by the band Saturday's Tinitus[2] and all the diligent people who took the time for proof-reading of this thesis.

Additional acknowledgment goes to all the artists of the songs and videos that have been used in this thesis. All these works were released under a creative commons license and they can be downloaded from *http://www.jamendo.com* if not indicated otherwise:

The song "Analcolico" by Millionaire Blonde, 2009 (cc-by-nc-sa), the song "Awesome Lie" by Stefano Mocini, 2011 (cc-by-nc-sa), the movie "Big Buck Bunny" directed by Sacha Goedegebure, 2008 (cc-by)[3], the song "Blue" by Susan Rom, 2009 (cc-by-sa), the song "Chuchelo" by Nastya Yasnaya, 2011 (cc-by), the song "Dorme Mariana" by Haroldo Torrecilha, 2007 (cc-by-sa), the song "Flood" by Tenpenny Joke, 2009 (cc-by-sa), the movie "Oceania" directed by Harpreet Dehal, 2008 (cc-by-nc-sa)[4], the song "Open Fields Blues" by Hiiragi Fukuda, 2011 (cc-by-nc-sa)[5], the song "Rumba Alemana" by El Perez, 2009 (cc-by), the song "Victory" by Alexander Blu, 2005 (cc-by-sa), the song "Warrior" by Blue Canoe Records, 2009 (cc-by-nc-sa).

[1] http://www.troxstudios.ch/
[2] http://www.saturdays-tinitus.de/
[3] http://www.bigbuckbunny.org/
[4] http://www.hdehal.com/filmandvideo.php
[5] http://freemusicarchive.org/music/hiiragi_fukuda/

Contents

List of Figures

List of Tables

1

Introduction

1.1 Motivation

Music constitutes an essential part of our life. It is a crucial part of visual media such as movies, video games and advertisements. With the dawn of the digital age accessibility and distributability of multimedia has increased and with it its value for society. More than 60% of people say they could not live without music[1]. Out of the pure producer-consumer relationship a remix culture has emerged [Les08]. Platforms like Youtube[2] reduced the distribution cost for an artist to zero and organizations like Creative Commons[3] provide a legal foundation for this movement. In the future media will very likely change further from a static consumer good to a resource that is available to everyone to build upon. This change in media production requires a new generation of tools, that allow experts and amateurs to efficiently process and change existing media works and which enable them to express themselves.

Inspired by the recent advancements in computer graphics, specifically animation, texture synthesis and, image and video retargeting, it is our goal to adapt these techniques to create novel tools to scale, edit and reshuffle existing music works. For example, in image retargeting, a given image can be rescaled to a new image with a different aspect ratio while the relevant image content is preserved and free of distortions. With the same principle the duration of a piece of music could be enlarged or shortened in a content aware manner.

Since the appearance of the computer many tools to synthesize and edit music have been developed. Synthesis focuses on additive or subtractive methods using synthetic signals or recorded samples. Editing focuses primarily on filtering and timescale-pitch modifications (cf. Section

[1] *Youth and Music Survey by Human Capitals for Marrakesh Records*, 2009, England, UK.
[2] http://www.youtube.com/
[3] http://creativecommons.org/

2.1.1). Nevertheless only little efforts have been made to develop efficient automatic editing solutions. Using today's tools music retargeting remains a manual iterative process that has to be performed by a skilled user.

1.2 Contribution

All the retargeting methods from computer graphics are in essence signal processing techniques and it should be possible to adapt them from two or three dimensional signals to music, a one dimensional signal. At first sight audio data appears to be less complex, but it is not trivial to handle at all. Audio signals are very high frequency, changing small numbers of samples can introduce annoying artifacts. Human listeners are very sensitive to discontinuities and changes in tempo. It is difficult to capture and manipulate the acoustically relevant structures, which are not local in nature but spread over several seconds of a song. Humans perceive rhythm, chords or melody as single objects. Also the large scale structure, the musical form, e.g. alternations between verse and chorus, has to be considered.

These are the main challenges of our goal: Meeting the desired duration, finding transitions that preserve the local and global properties of the music, preventing obvious repetitions and providing high-level user control. We investigated many different approaches such as incremental texture synthesis and seam carving. Our proposed method is based on a graph representation inspired by the concept of motion graphs from computer animation. Given a certain target length and additional constraints, such as song structure or importance, the proposed method performs a global optimization to find an assembly of slices of the input music that fulfills the constraints with the best possible quality. Low and high-level features are extracted from the audio signal to build a graph, which encodes the constraints and potential forward and backward jumps. The desired music is synthesized using a k-stops shortest path algorithm using dynamic programming. The algorithm runs in real time and creates results with imperceptible transitions.

Our method offers a wide range of applications in audio production, video editing and video game production. We present different applications such as shortening and enlarging the duration (up to infinity), music structure reshuffling, removal of content (e.g. vocals) and video editing with audio alignment.

The remainder of the thesis is organized as follows. In Chapter 2 we present related work from the audio and graphics community. Chapter 3 gives a brief overview of our algorithm. Chapter 4 describes the first part of the pipeline, the extraction of features and Chapter 5 the following synthesis part. The results are presented in Chapter 6. We finish with the conclusion and the outlook in Chapter 7.

2

Related work

This chapter reviews the existing work related to music retargeting from the audio and computer graphics community. The reviewed topics range from signal processing to image resizing in computer graphics.

2.1 Audio processing

2.1.1 Music retargeting

The goal of a retargeting process is the adaption of the length of a signal to a desired output length. For instance, music retargeting reduces a 3-minute music to a version of 1 minute.

The most trivial approach to perform a retargeting would be to crop at the start and the end of the song. But unlike in image processing, where the borders of images mostly contain data of little importance, in music these parts are very distinctive elements and should be preserved. An example of a very recognizable introduction is the song "Money" by Pink Floyd[1], which contains clinking of coins and a ringing cash register. An example for a distinctive end can be found in the track "In the Stone" by Earth Wind & Fire[2].

A more involved approach can be performed by an artist using conventional audio tools. The artist chooses and cuts out segments from the original song. The remaining segments are aligned to form the new song. The boundaries between the segments are typically blended to restore continuity. High quality results can be obtained but a lot of time and a highly skilled user is

[1] http://www.youtube.com/watch?v=LjMWwf8RtTQ
[2] http://www.youtube.com/watch?v=fbPxUxNtVR8

required to find out appropriate cut positions.

A fully automatic and well known technique in the music community is the audio timescale-pitch modification [Lar02]. Let's assume we want to increase the playback speed of a song to reduce its duration. If we just resample the signal, then the resulting song has a different pitch (e.g. voices sound like Mickey Mouse). A timescale-pitch modification can be used to change pitch or playback speed independent form each other. This allows not only temporal changes but also the correction of wrong notes in recordings. Today, pitch adjustment is a standard post-processing tool in the production of pop music.

Timescale-pitch modification can be performed in time or frequency space:

- The **time space** family of methods is called Overlap-Add (OLA) [Ver00]. A song is cut into small overlapping slices, typically 100 milliseconds. These slices are removed or copied and then merged together into the output song, depending on whether an enlargement or shortening is required. It is crucial that the transitions between the slices do not create distortions. Many methods, that are based on this principle, have been proposed in the past decade. Well known variations are Pitch Synchronous Overlap-Add (PSOLA) [KK97] and Waveform Similarity Based Overlap-Add (WSOLA) [GL08]. Most of the OLA methods differ only in their strategies how slices are selected and aligned.
- The **frequency space** approach is called Phase Vocoder [LD99]. The song is transformed into the frequency domain using a short-time Fourier transform (STFT). The spectral coefficients can be stretched to shift the pitch. The transformation back into the time domain can be done using a different sampling rate to change the duration.

While timescale-pitch modification can produce reasonable results for small scaling factors (up to about +/-20%), it leads to important artifacts (hall, echo, drift, stretch distortion) for higher factors [LD99].

Paul's Extreme Sound Sketch[3] is an implementation of a timescale-pitch method which can handle large scaling factors (up to 10^{18} times the original length). It tries to minimize the amount of distortion, but the result has only little resemblance with the original song, as it gets transformed into a noisy sound texture. This is not what we want to achieve: Our goal is to preserve the original sound. Please refer to Section 6.7 and the supplementary material for example results obtained by this method.

Liu et al. [LWB+11] builds upon WSOLA time scaling and proposes a method that enables bigger duration changes than pure OLA approaches. It introduces the concept of a stretch-resistance measure for music. Based on this measure some parts of the song are stretched intensively, other parts only lightly (e.g. parts that contain vocals). For extreme shortening request they also remove repetitive sections by segmenting the song and removing repetitions using a greedy algorithm. While the methods outperforms previous methods, it does not address the fundamental problems of the OLA technique (echo, distortions, etc.) and the segment removal is based on heuristics and extensive blending. It is therefor unable to find an optimal solution.

The most recent approaches to music retargeting are purely based on removing or repeating slices of the original song. *Wenger et al.* [WM11] proposes a method which is based on short-

[3] http://hypermammut.sourceforge.net/paulstretch/

time Fourier transform (STFT) based features. From these features a multi-resolution self-similarity matrix is computed which is used to find a set of good jump position candidates. The output song with the desired length and a minimal perceptual error is found by applying a heuristic best-first search. The method finds reasonable results for certain music genres but there is no guarantee that it obtains the globally optimal solution of the proposed cost function. In addition it doesn't make use of any knowledge from music theory. The beat and high-level song structures are not preserved.

The Echo Nest[4] audio platform provides a web service for music retrieval and analysis. *Earworm* is an example application that is build upon the Echo Nest API. It uses timbre and pitch features which are used to compute self-similarity matrices. Different heuristics are used to extract a set of good loops. Loops are sorted by length and selected in a greedy manner until a length close to the desired one is reached. To meet the start and end constraints of the new song, a graph is formed out of the jumps positions and A* path searches [HNR68] are performed to connect the initial loop solution to the desired start and end position. The algorithm often fails to find a solution close to the desired length (e.g. a request to resize the song "Open Fields Blues" by Hiiragi Fukuda from 162 to 121 seconds could only be resized to 127 seconds, which is a duration error of 6 seconds) and sometimes fails to find a solution at all. See Section 6.7 for a comparison of *Earworm* with our proposed method.

2.1.2 Music summarization

A different form of retargeting is the music summary, also known as music thumbnail. Music summarization focuses on extracting the most representative clip in a music track. Such clips usually consist out of the main melody, the main theme or the chorus.

Music summaries are useful to efficiently browse a large collection of music files and they are used as a compact representation for music retrieval systems. An example usage is Amazon.com's music store where customers can listen to 30-second-excerpts of songs before purchasing them.

Most approaches try to extract the most frequently appearing sequences in a song, which can be achieved using clustering based on a hidden Markov model (HMM) [LC00] or the analysis of self-similarity of Mel-frequency cepstral coeficients (MFCC) [CF02] or chroma features [BW05]. *Xu et al.* [XZT02] uses temporal, spectral and cepstral features in combination with a clustering approach to find the most frequent parts. Other approaches generalize the pure repetition based approach by defining a measure of audio saliency which is used to extract the most rememberable parts. Saliency can be estimated using occurrence frequency, energy information and positional weighting [LZ03].

Music summarization works well for their intended use case of browsing and retrieval, but these methods do not provide a song of a desired length, nor do they enforce continuity between sequences. In addition the beginning and end of a clip is abrupt.

[4] http://the.echonest.com/

2.1.3 Sound textures

In analogy to the methods to synthesize textures in computer graphics (e.g. [Ash01]), the concept of sound or audio textures has been defined. [SERlG06, Sch11] give a recent overview of the efforts that have been made in the area of sound texture synthesis.

Early methods focused on analyzing the signal of a sound texture, learning its parameters and then synthesizing new data by a granular additive or subtractive audio synthesis technique. More recent approaches are example based and apply tiling and stitching to synthesize new sounds. *Parker et al.* [PB04] adapts algorithms from image texture synthesis, namely *image quilting* and *chaos mosaic*. *Lu et al.* [LWZ04] creates audio textures by splitting the input into slices. Based on the similarity between slices a transition probability from one slice to another is computed. This enables a probabilistic synthesis of audio textures.

Interesting results can be obtained for repeating sounds (such as rain, fire and wind) and short ambient music, but these methods clearly fail for long sequence of structured music.

We investigated the adaptation of texture synthesis approaches for music synthesis. We adapted the paper *Synthesizing natural textures* [Ash01] by interpreting the music signal as a one-dimensional texture. The approach works in principle, but this purely local greedy approach often leads to undesired repetitions. The synthesis can easily end up in dead ends due to the low variability of the one-dimensional neighborhood, which leads to transitions of very bad quality. It also cannot maintain the music structure.

2.2 Computer graphics

2.2.1 Music synthesis in computer graphics

Most of the existing works about audio in the computer graphics community focus on music synthesis using physical models, for example generating sound effects for colliding objects or liquid sounds.

Chadwick et al. [CJ11] described a method for synthesizing plausible fire sounds that are synchronized with a physically-based fire simulation. *Zheng et al.* [ZJ10] proposed a physically-based algorithm for synthesizing sounds synchronized with brittle fracture animations. *Cardle et al.* [CBBJR03] uses existing animation sounds to train a statistical model, which is used to synthesize sounds that match a new animation. This wavelet-based technique allows to create effects and ambiences at a coarser level than pure physically-based approaches.

2.2.2 Retargeting in computer graphics

Image retargeting aims at resizing an input image to a given resolution with a changed aspect ratio and without distortion. This research topic emerged with the work of [AS07], which presented a method called seam carving. The main idea consists of removing the seams with the least amount of energy. Since then numerous variants have been proposed (cf. [RGSS10]

for a review and comparison).

Applying the seam carving algorithm, or any of its variants, would be the most evident way to resize music. We implemented this technique and tried several kinds of music, but it did not work well in practice. To choose the position of the seams to remove, a saliency measure for music is required. We evaluated saliency measures based on occurrence frequency [LZ03] or novelty [Foo00], but they are not reliable and robust enough yet. Additional challenges are the preservation of the rhythm and the continuity after seam removal. Preserving the beat or the bar requires very coarse carving operations, i.e. large seams, which makes retargeting rigid and often creates low quality transitions.

Lefebvre et al. [LHL10] applies the concept of retargeting to synthesizing architectural textures given an example image. Horizontal and vertical slices are extracted from the input texture to assemble new textures. A graph of all possible solutions of the desired length and constraints is set up. The resulting texture is found by a shortest path search where the path length corresponds to the cost of the assembly. In principle this retargeting approach could also be applied to music, but audio features have a much smaller scale than architectural elements and therefore the proposed optimization method would be computationally unfeasible. In addition the synthesis lacks any modeling of high-level structures, which would lead to undesired repetitions in the music case.

Retargeting can also be applied to a full 3D model of buildings [LCOZ+11]. Depending on user annotations, facade elements of a building are either repeated, scaled or preserved to create new buildings. For audio application, scaling would be mapped to some OLA method, which are known to create artifacts for big scaling factors (cf. Section 2.1.1) and obvious and frequent repetitions are not desirable in music.

Little work has been done to perform temporal retargeting of media. *Schödl et al.* [SSSE00] introduces a new medium called *video texture*. Given a video, the algorithm finds a set of loops that forms a video close to a desired length. Our self-similarity based music analysis is inspired by the video analysis method of this paper, but the synthesis method is not well suited for our requirements. The loop representation of the video is optimized towards a stochastic synthesis. They also propose a non-stochastic optimization algorithm that tries to schedule an optimal set of loops, but their method can not make sure that a certain duration is reached because the result depends on the variability of the available loop durations. In addition it is very likely that repetitions of the same frames occur very often, which is undesirable for music.

3

System overview

This chapter gives an overview of the proposed algorithm and its implementation. First the abstract synthesis interface is presented, secondly the algorithm's main building blocks and their dependencies are explained.

Our algorithm has one general synthesis interface, shown in Listing 3.1, which covers all the proposed use cases. The input consists of the audio samples of the original song (inputSong), its segmentation (inputSegmentation) and an optional importance function (importanceFunction). The importance function assigns a scalar value to each sample of the input song. There are many options for valid input segmentations; just one trivial segment from the beginning to the end of the song, an automatically extracted structural segmentation or a user supplied custom segmentation, e.g. a segmentation into vocal and instrumental parts. The desired output is defined by the start (startPosition) and the end (endPosition) positions of the new song in the original song and the segmentation of the output song (outputSegmentation). The interface returns the audio signal of the new song (outputSong) and the cost of the optimization procedure, which indicates the quality of the result.

Listing 3.1: *Synthesis interface in MATLAB*

```
[outputSong, cost] = function synthesize(inputSong, ...
                                         inputSegmentation, ...
                                         importanceFunction, ...
                                         startPosition, ...
                                         endPosition, ...
                                         outputSegmentation)
```

To illustrate the usage of the interface lets have a look at an example; content aware retargeting to half the original duration. The input segmentation is set to the automatically extracted song structure and the importance function to a constant value (e.g. 1), as we do not prefer any part

of the song to another. The start and the end positions are set to the original start and end positions. The desired output segmentation is the same sequence of labels as the input, but all segment boundaries are scaled by a factor of 0.5. The synthesis returns a song with the same structure as the input, but with half the original duration.

Figure 3.1 shows the flowchart of our algorithm. The pipeline consists of two major components: the analysis and the synthesis parts.

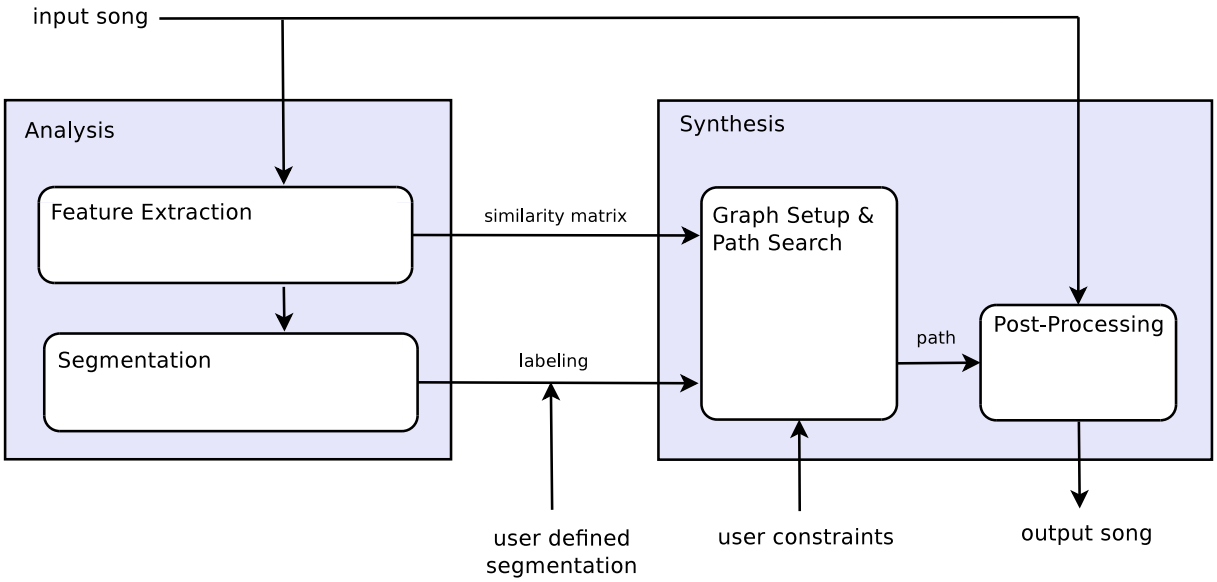


Figure 3.1: *Flowchart of the proposed algorithm*

In the **analysis** component low and high-level features of the song are extracted. From the input samples of the song first, a beat aligned self-similarity matrix (**feature extraction**) and second the structure, a labeled **segmentation**, is computed. Both analysis products are then passed to the **synthesis** component. To generate the desired music a graph representation of the input song is set up (**graph setup**) and a path that satisfies the user's constraints is searched (**path search**). The resulting path encodes how parts of the original song have to be aligned in order to create the desired output song. **Post-processing** assembles the path directly to an audio signal or processes it further to improve the perceptual quality of the transitions or optimize the precision of the duration.

The analysis of 4 minutes of music takes about 50 seconds using our MATLAB implementation. The analysis component is independent of the synthesis parameters and for that reason the products of the analysis can be reused for different synthesis requests. The path search is implemented in C++, which allows to perform the music synthesis in less than a second and therefore permits real-time user interaction.

4

Music analysis

This chapter explains how we analyze the audio signal to extract a meaningful representation. Section 4.1 describes the extraction of low-level features with the goal to create a self-similarity matrix that can be used to find perceptually and temporally consistent jumps inside a song. Section 4.2 builds upon the low-level features to create a high-level classification of the song, a structural segmentation. The analysis is performed on a mono audio signal with a bit rate of 44.100 samples per second. Silent parts at the start and end of the song are trimmed. The silence is detected by computing the energy of the audio signal inside a sliding window. If the energy is below a certain threshold the window is classified as silence.

Although the analysis is performed on a mono signal any number of audio channels can be synthesized. Section 5.5 explains this in more detail.

4.1 Feature extraction

Music is difficult to analyze and manipulate because of its high data rate (up to 4.4 Mbps per audio channel on a Blu-ray disk), the redundancy in the signal and the way humans perceive it. The human hearing is not local in nature but instead based on higher level objects (like beat or melody) which are spread over a certain time frame. It is therefor important to extract relevant data using a compact description. This process is known as feature extraction. For image processing feature extraction techniques like edge detection [FP02] and SIFT [Low04] are widely used to extract and/or describe reliable features in images. A similar concept to manage and extract information from audio signals exists. Three major groups of audio feature descriptors have been developed. They describe the chromatic, timbral or rhythmic characteristics of a window of an audio signal [PMK10].

Chroma features, also called *Pitch Class Profile* or *Chromagram* describe the harmonic content of a window. The descriptor is a 12 dimensional vector; each value represents one of the pitch classes in Western music notation (C, C#, D, D#, E, F, F#, G, G#, A, A#, B). A normalized chroma vector describes how the signal's spectral energy is distributed among the 12 pitch classes, but it ignores octave information [Góm06].

Timbral features are motivated by the human hearing and model how sound sources are perceived. They encode the relative levels of sound at critical bands and their temporal evolution. Popular timbre descriptors are the *Constant Q transform* [BP92] and the *Mel-frequency cepstral coefficients* (MFCC) [Mer76]. The MFCC is one of the most popular descriptors in audio analysis [PMK10]. Originally developed for speech recognition, it has been shown that it also performs well on music [Log00]. MFCC can be represented by a vector of size N (typically 40 dimensions) and it is computed as follows. The logarithm of the power spectrum of the signal is computed using the Fourier transform. The resulting powers are then mapped into N non-linear Mel scale bins. The term "Mel" comes from melody and its scale transforms the frequency f so that the pitches are equidistant for a human listener [O'S87]:

$$m = 2595 \log_{10} \left(1 + \frac{f}{700} \right) \tag{4.1}$$

The MFCCs are obtained by the discrete cosine transform (DCT) of the N uniformly distributed Mel-scale bands m:

$$MFCC(k) = \sum_{m=0}^{N-1} E(m) \cos \left(\frac{\pi(2m+1)k}{2N} \right) \tag{4.2}$$

where $E(m)$ is the energy of the frequency band m. Most implementations discard the first coefficient and replace it by the logarithm of the energy of the frame.

The third group of descriptors try to capture tempo and beat. They are often represented in a spectrogram-like representation called *rhythmogram* [Jen07], *tempogram* [CKDH00] or *beat spectrum* [FU01]. Most methods first extract the onsets of each note using the so called *onset accent curve*. In the second step the onset curve is analyzed for periodic structures using autocorrelation or STFT based methods [GM09]. Rhythm descriptors are well suited to describe a full song or large parts of it, e.g. for music retrieval tasks, but their discriminative power is limited when a self-similarity analysis of small windows has to be performed.

We evaluated the different descriptor families and experiments have shown that the MFCC descriptor performs best for different music genres. The descriptor is computed from an STFT which typically has a fixed step and window size. These descriptors would not capture the rhythm of the song and in the synthesis step transitions would not preserve the beat. We therefore propose a dynamic window size based on the beat length. The length of beat windows generally varies between 250 milliseconds and 1.5 seconds. As an analogy to pixels in image processing, we call these beat segments **"bixels"** (a contraction of the words beat and pixel) to avoid confusion with the segments introduced in Section 4.2; "bixels" are the smallest working unit in our pipeline. Accurate beat information is obtained using BeatRoot, an implementation of the beat tracking algorithm by *Dixon* [Dix07]. For most songs the beat can not be tracked at the beginning and end parts and beat extrapolation is required to obtain a complete bixel representation.

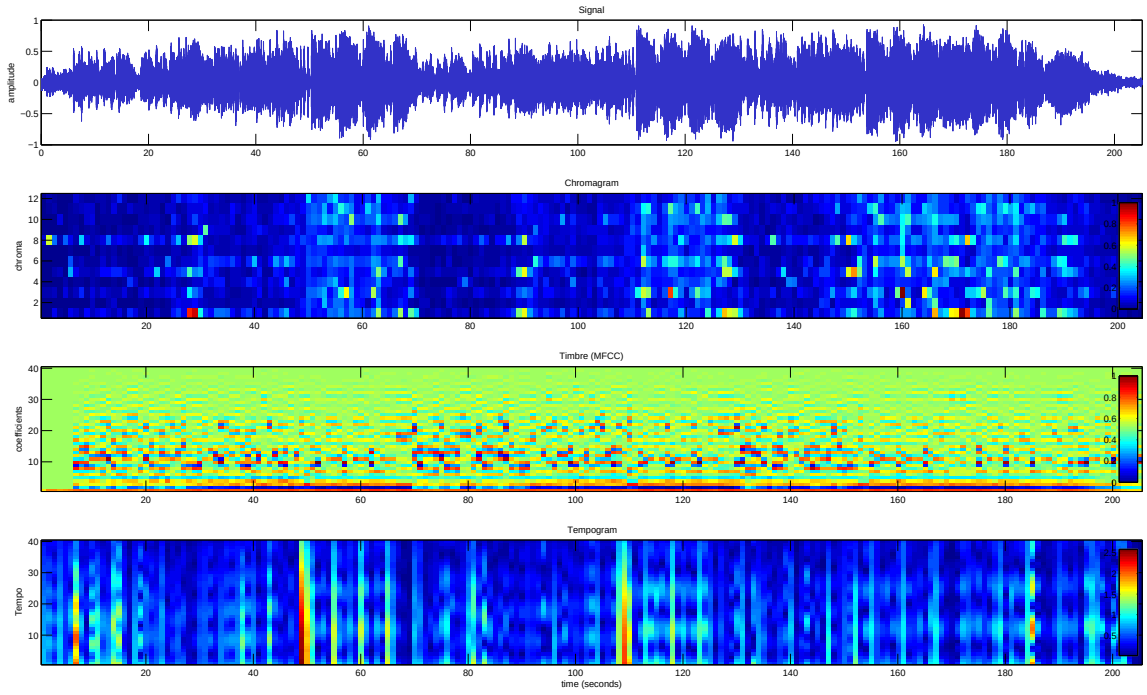


Figure 4.1: *Different feature descriptors of the song "Lucy In The Sky With Diamonds" by The Beatles. The first plot shows the signal of the song. For each descriptor matrix (from top to bottom: chroma, timbre, and tempo) a fixed window size of one second was used.*

Once the MFCC are computed for each bixel, they can be used to compare the associated parts of the music and thus find the perceptually similar parts. To compute the distance between all the MFCC, we tested several distance measures including cosine, $L1$, $L2$ and different correlations. Experiments have shown that the most reliable results were obtained by Spearman's rank correlation [MW10] in terms of discriminative power and robustness.

Spearman's rank correlation coefficient measures the statistical dependence between two vectors. It makes no assumption about the probability distribution of the measurements and it is robust against outliers as the rank of a value inside the measurement is compared and not the value itself. The measure has a result between 0 and +1 if large values of one variable are associated with large values of the other and small with small. It has a result between 0 and -1 if large values of one variable are associated with small values of the other.

Spearman's rank correlation is computed as follows:

$$Corr(i,j) = 1 - \frac{6 \sum_{k=1}^{n} \left(rank(l_i(k)) - rank(l_j(k))\right)^2}{n(n^2-1)}, \tag{4.3}$$

where n is the number of dimensions of an observation l_i (i.e. the number of coefficients of our descriptor) and $rank()$ is a function that returns the position of the value in the sorted list of all the values of the appendent observation. $l_i(k)$ denotes the k^{th} coefficient of the descriptor l_i.

The correlation of all descriptors $Corr(i,j)$ forms a matrix with values in the range of $[-1,1]$,

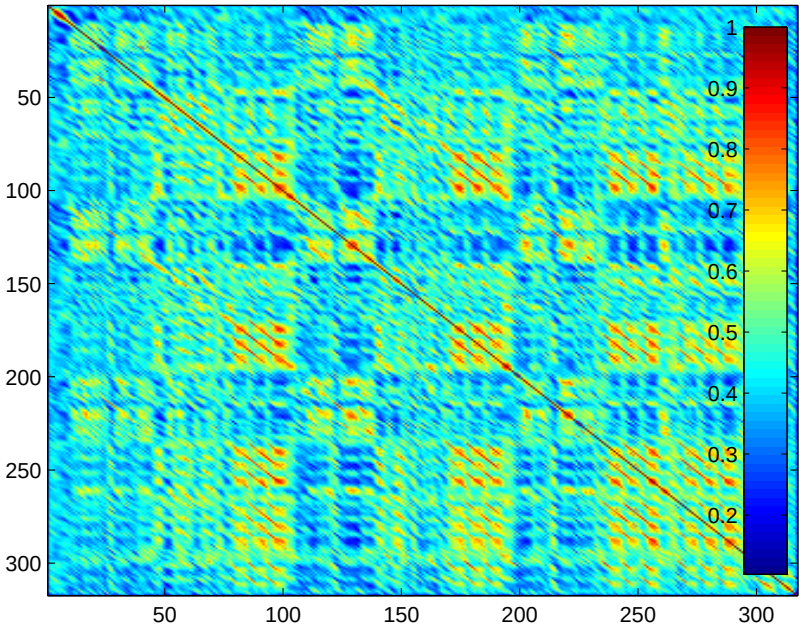


Figure 4.2: *Self-similarity matrix of the song "Lucy In The Sky With Diamonds" by The Beatles. Both dimensions represent time. Repeating square patterns and lines parallel to the diagonal become visible. These structures correspond to segments with similar content.*

which has to be converted into the similarity matrix $S(i, j) \in [0, 1]$:

$$S(i, j) = \frac{Corr(i, j) + 1}{2} \tag{4.4}$$

The distances are now stored in a beat-aligned self-similarity matrix where the entry $S(i, j)$ contains the similarity measure between the beats i and j of the input song. The resulting matrix captures the perceptual similarity between two beats but does not contain any temporal information. To incorporate the dynamics of the music, we consider not only the current beat but also the temporally adjacent beats as follows:

$$S'(i, j) = \sum_{t=-m}^{m} w_t S(i + t, j + t) \tag{4.5}$$

where the weights w_t are the normalized binomial coefficients and we set $m = 2$ for all the results presented in this thesis. A similar approach was used in [SSSE00] for video similarity. The value of $S'(i, j)$ is small when the i^{th} and j^{th} beats are both perceptively and temporally consistent. S' is normalized to the interval $[0, 1]$.

4.2 Segmentation

Assume we want to create a song with half of the original duration. In a straight forward approach we would apply our algorithm on all bixels of the whole song. It is very likely that the result that we would obtain uses only one or two jumps, because our shortest path based synthesis tries to minimize the number of jumps used. Such a solution would use one jump from the first chorus part to the second chorus part and the segments in between, verse or bridge, would be removed. For many use cases this behavior must be avoided. Instead we want to preserve the high level structure of a song. Having a labeled partitioning of the song allows us to scale each part individually. Figure 5.3 shows a comparison of labeled and unlabeled retargeting. In addition it allows us to create a new structure for the output song by removing or reordering segments (cf. Section 6.4 and 6.5).

We define the term **"segmentation"** as follows. A segmentation defined by a sequence of time stamps divides a song into homogeneous parts which can be identified by changes of melody, theme, instrumentation or vocals. Equally valid criteria are singing vs. instrumental parts or the content of the lyrics. Each part is assigned with a label. The labels group similar parts into a category, e.g. bridge, chorus and verse parts. The principal issue of music segmentation is the ambiguity of the task. Segmentations of the same song performed by different experts can lead to large variations in the results [PLR07]. It is therefore difficult to acquire ground truth data and evaluate the performance of different methods.

Many methods for music segmentation have been proposed. *Paulus et al.* [PMK10] gives a recent overview of popular algorithms. There are three principal strategies: (1) repetition based approaches try to identify repeating patterns e.g. by extracting parallel lines using image processing techniques on the self-similarity matrix of a song, (2) novelty-based approaches try to detect transitions between contrasting parts and (3) homogeneity-based methods group intervals where certain music features are consistent.

We have chosen a segmentation strategy which uses a combination of a novelty and a homogeneity-based approach. It stands out for its robustness, small number of parameters and ease of implementation. The input of the segmentation procedure is the self-similarity matrix obtained in Section 4.1. The matrix is filtered using a 5 by 5 Gaussian kernel to remove low frequency noise as we are only interested in large scale structures that span at least over several seconds.

The extraction of the segment boundaries is based on the the concept of the *novelty score* as proposed by [Foo00]. We want to detect the discontinuities in the similarity, because these regions tend to indicate the beginning or end of a structure. This operation can be considered an edge detection [FP02] operation on music. The detection kernel is a 2 by 2 checkerboard pattern, which is scaled to the desired size and weighted by a two dimensional Gaussian function. An efficient way to setup the checkerboard pattern is to multiply the 2 by 2 pattern using the Kronecker product with an identity matrix of half the desired size. Equation (4.6) shows the example for a kernel size of 4.

$$K(m,n) = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{pmatrix} \tag{4.6}$$

Figure 4.3 shows an example kernel of size 64 by 64 including Gaussian weighting. The novelty score is the response of this kernel when it is convolved along the diagonal of the self-similarity matrix:

$$N(i) = \sum_{m=-L/2}^{L/2} \sum_{n=-L/2}^{L/2} K(m,n)G(m,n,\sigma)S'(i+m,i+n), \tag{4.7}$$

where L is the size of the kernel and G is a normalized 2D Gaussian of size L with $\sigma = L/3$.

The peaks of the novelty score are good indicators for changes in timbre or instrumentation. The default kernel size in our pipeline is 96x96, which corresponds to a detector of structures of about 20 seconds duration. We select all the local extrema of the normalized novelty score that lie above a threshold of 0.1 as our segment boundaries. The first image in Figure 4.4 shows an example of a novelty scores with different kernel sizes.

Once the boundaries are extracted, the bixels inside each segment are averaged to obtain one descriptor for each segment. These new descriptors must be grouped based on their similarity. Each group of segments defines a label. This can be achieved using clustering, for example using the k-means algorithm [Mac67], where K corresponds to the number of labels. Typical values for K are between 1 and 5 and it can be adjusted interactively by the user if necessary. In our implementation spectral clustering by normalized cuts [SM00] is used. Experiments have shown that it outperforms k-means, as clusters are not restricted to high dimensional spheres but can have arbitrary shapes. The obtained labels tend to segment the songs into semantic structures such as chorus, bridge and verse. Figure 4.4 shows a segmentation created with our method along with a ground truth segmentation for comparison.

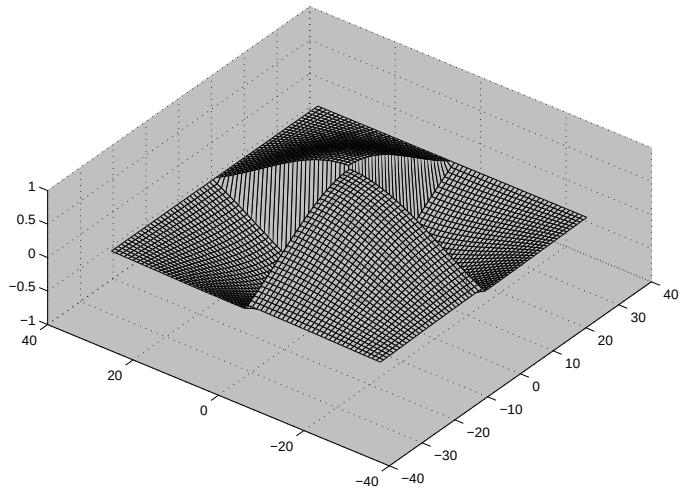


Figure 4.3: *Novelty score kernel (64x64) plotted as a surface in 3D space.*

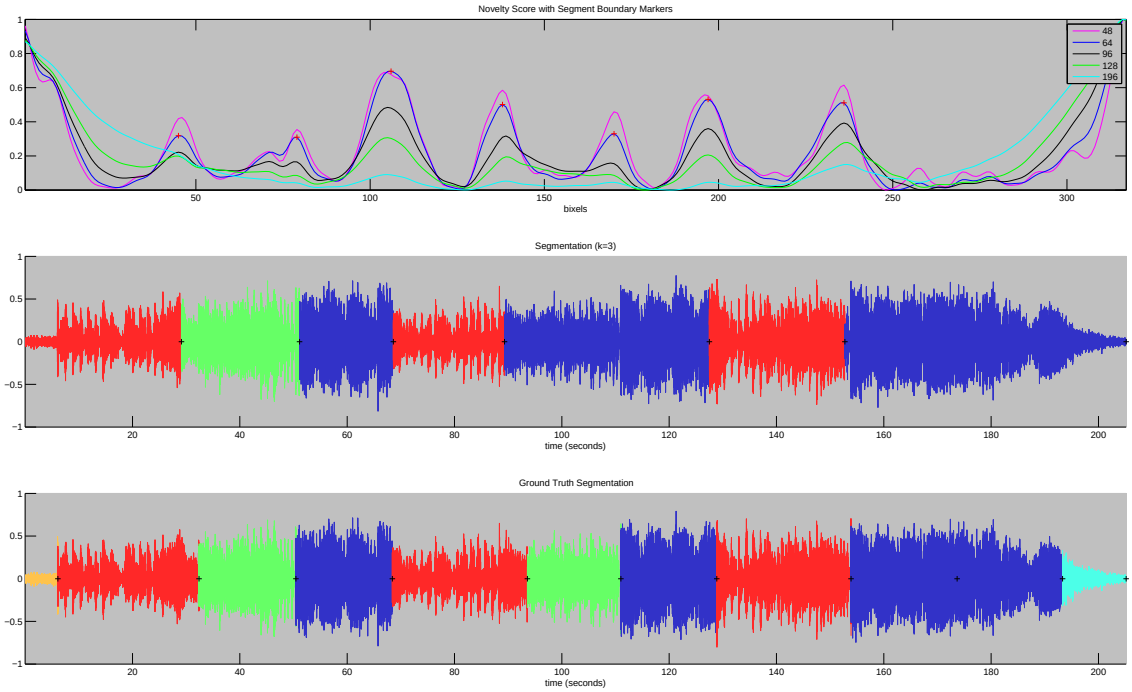


Figure 4.4: *Segmentation of the song "Lucy In The Sky With Diamonds" by The Beatles. The three plots show the following:*
(1) Novelty scores computed from the similarity matrix in Figure 4.2 using different kernel sizes. The red crosses mark the local maxima that define the extracted segment boundaries.
(2) Audio signal of the song with the labeled segmentation created by our method.
(3) Audio signal with the ground truth segmentation provided by [PLR07].

In this example the novelty kernel size has only minor influence on the position of the local maxima. The intro and outro segment borders could be found using a kernel size of 48 or smaller. With the exception of one central segment the labeling is equivalent to the ground truth. Some segment boundaries are slightly shifted compared to the ground truth, which is probably caused by the coarse resolution (bixels) of our similarity matrix.

5

Synthesis

The previous chapter focused on the extraction of low and high-level features. In this chapter we will use these features to synthesize new music that satisfies a given set of constraints. Our approach is inspired by the concept of motion graphs, a popular approach in computer graphics to synthesize animation given a corpus of example animations and new constraints.

This chapter starts by defining the mathematical formulation of music retargeting as an optimization problem. Then we discuss the relevant optimization methods and present our dynamic programming-based technique. In Section 5.4 we expand the technique by adding high-level control using the concept of motion graphs. In the final Section 5.5 we explain how the solution of the optimization step is transformed into the final output signal.

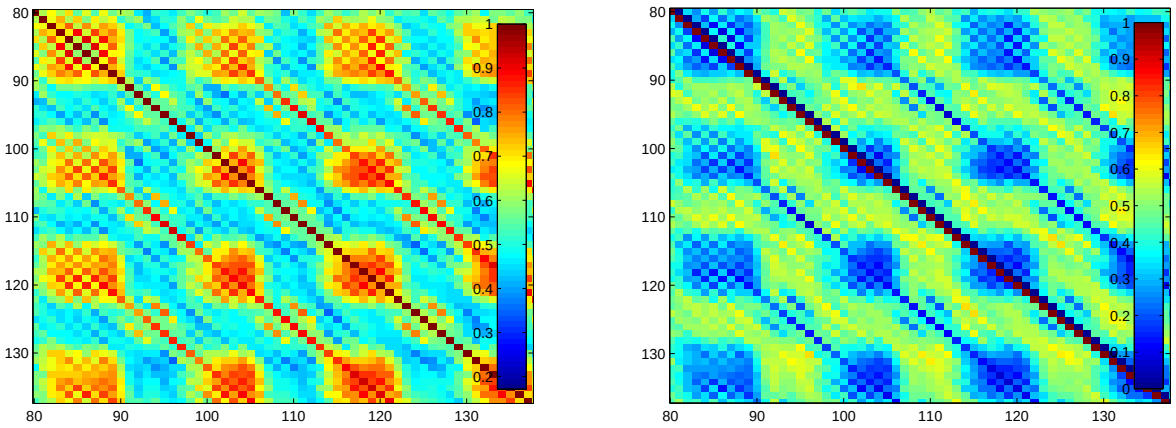
5.1 Problem formulation

Given a song with N bixels (cf. Section 4.1) we want to create a new song with the new length of M bixels, which starts with bixel s and ends with bixel t. The start and end positions can be the first and last bixels of the input song to perform retargeting or any two bixels selected by the user, e.g. to create a loop. The input music is modeled as a *complete directed graph* where each node represents a bixel. Nodes can be associated with an importance value. Edges define possible subsequent bixel and are weighted with the dissimilarity (distance) between the naturally following bixel and the target bixels. We define the term **"jump"** as a transition (edge in the graph) between two nodes (bixels) that do not naturally follow in the input music. It points to any other bixel in the music and makes shortenings or enlargements possible. Jumps in our figures (e.g. Figure 5.2) are depicted as arcs with a circle at their start positions. Jumps that go forward in time are colored purple, backward jumps are colored in green.

The graph's adjacency or dissimilarity matrix D can be computed directly from the self-similarity matrix S' that we obtained in Section 4.1 as follows:

$$D(i, j) = 1 - S'(i + 1, j) \tag{5.1}$$

$D(i, j)$ encodes the cost given the bixel i to use the bixel j as the next adjacent bixel. It encodes the cost to violate the audio continuity. The bixels that would naturally follow in the input song $D(i, i + 1)$ have zero cost as their similarity is one (perfectly similar). The diagonal elements of this matrix $D(i, i)$ are reset to 1 (highest possible value) to remove the possibility to repeat the same bixel without at least one intermediate different bixel.



(a) similarity (b) adjacency

Figure 5.1: *Cutout of the similarity matrix* (a) *and the resulting adjacency matrix* (b) *(with reset diagonal) of the song "Flood" by Tenpenny Joke.*

Given such a graph, the goal is to select a path $\mathbf{l} = \{l_i | i = 1 \dots M\}$ comprising bixels l_i to obtain a retargeted song of the desired length M. Additionally, we constrain the start and end nodes, such that $l_1 = l_s$ and $l_M = l_t$, respectively.

We define the cost of a path based on the distance value in the adjacency matrix and the importance of a node:

$$E(\mathbf{l}) = \lambda \sum_{i=1}^{M} I(l_i) + (1 - \lambda) \sum_{i=1}^{M-1} D(l_i, l_{i+1}), \tag{5.2}$$

where the function $I(l_i)$ encodes the inverse importance of the bixel l_i, the function $D(l_i, l_j)$ represents the dissimilarity between the two bixels l_i and l_j, and the constant λ balances the influence of the two terms.

Finally the problem is formulated as a minimization of the cost function E:

$$\mathbf{l}^\star = \arg\min_{\mathbf{l}} E(\mathbf{l}). \tag{5.3}$$

5.2 Optimization methods

The optimization problem defined in the previous section has a solution space that is exponential in the number of bixels N. It can be seen from two different views, either as a multi-label graph cut problem or a shortest path search. It is crucial that the optimization method is able to provide solutions that include loops. Without loops music duration enlargements are not possible.

In the *multi-label graph cut* view, the original bixels correspond to the labels and the positions of the bixels in the new song correspond to the graph nodes that have to be labeled. The cost function (equation 5.2) can be interpreted as a combination of data and smoothness terms, where the importance corresponds to the data term and the distance to the smoothness term. The problem is known to be NP hard, but approximations have been proposed, especially [BVZ01], a popular and efficient algorithm for image segmentation. However [BVZ01] assumes that the smoothness term is symmetric, which is not the case in our synthesis scenario, i.e. playing first bixel i and then bixel j gives a different audio perception than playing bixel j first and then bixel i.

The other view of the problem is the interpretation as a graph where the solution with minimal cost, which corresponds to perceptual error and importance, is a shortest path. The standard algorithm is Dijkstra [Dij55], but the number of nodes in the output path and thus the music duration can not be controlled. Efficient extensions to extract the K *shortest paths* with loops have been proposed [Epp98], which could be used to compute an increasing sequence of shortest paths (short in terms of cost) until we get the first path with the desired number of nodes. While appealing at first in the worst case millions of paths have to be computed to find the solution with the desired number of nodes.

Other extensions of the shortest path problem allow the enforcement of constraints such as the number of nodes [Zie07]. These so called *k-stops shortest path* algorithms search for a path with minimal cost, start and end at user defined nodes and visit exactly k nodes in total. Only a small number of k-stops shortest path algorithms have been proposed, but none of them can be applied to our setting. Such as by *Terrovitis et al.* [TBPM05], which is unfeasible for our purpose because it assumes a spatial/Euclidian space.

5.3 Proposed method

We now present our proposed method to solve the minimization problem defined in equation (5.3). Our method finds the globally optimal k-stops shortest path using dynamic programming. A similar approach has been used by [Vit67] to find the most likely path of length k through a graph of states governed by a Markov chain. The proposed solution supports loops, i.e. the same nodes can appear several times in a path.

We can observe that the importance term is not dependent on the order of the resulting path. This allows us to merge the importance I and the dissimilarity D into one cost matrix called W.

$$W(l_i, l_j) = \lambda I(l_j) + (1 - \lambda) D(l_i, l_j) \tag{5.4}$$

defines the cost of choosing to place bixel l_j after bixel l_i. The constant λ balances the influence of the two terms.

Let $C_s(i,k)$ be the cost of the minimum path from bixel s to bixel i with k-stops, which can be defined recursively:

$$C_s(i,k) = \min_{j \in \{1..N\}} \Big(C_s(j,k-1) + W(l_j, l_i) \Big) \tag{5.5}$$

The recursion has to be initialized with $C_s(i,1) = D(l_s) + W(l_s, l_i)$ and $C_s(t,M)$ will be the cost of the optimal path of length M with the start and end nodes l_s and l_t.

C_s can be computed efficiently in $O(N^2 M)$ using an array of size $N \times M$. Starting with the first column at $k = 1$ each row i of $C_s(i,k)$ is computed until $k = M$. For each result of $C_s(i,k)$ the previous segment l_{k-1}, which gave the minimum cost to reach the current segment i, is stored to be able to backtrack the final path later. After all array values are computed, the path is backtracked starting from $C_s(t,M)$. The previous bixel l_{k-1} is found by following the bixel index stored along with the cost of the bixel at length k. An example optimization result of a shortening with importance can be seen in Figure 5.2.

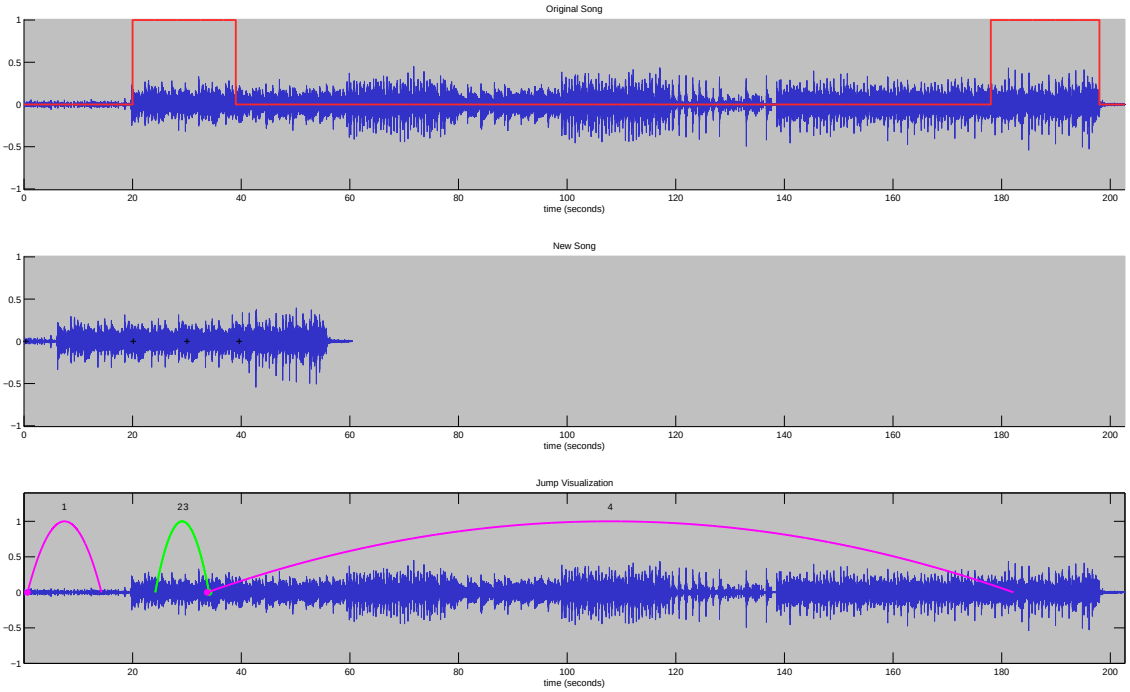


Figure 5.2: *Example synthesis where an importance function (red line in the first plot) is used to influence the output. The song "Analcolico" by Millionaire Blonde is resized from 3 minutes 28 seconds to 1 minute (second plot). The resulting song consists nearly entirely out of parts where the artist is singing "la la la" as requested by the importance. The third plot visualizes the jumps in the original song to generate the output song. Purple lines indicate forward and green backward jumps. The second jump is used twice as indicated by the sequence numbers on top of each jump.*

For some use cases it is desirable to relax the duration constraint and prefer the perceptually best solution within certain tolerance, e.g. +/-10%. This duration versus quality trade-off can

be achieved by computing solutions for all the different durations within the tolerance range and choosing the one with the smallest cost.

5.4 Constraints

With the optimization framework presented so far in Section 5.3 we are able to enlarge or shorten music and define start and end positions. The results have optimal transition points and the importance function can be used to influence the synthesis process. However the retargeting process is unaware of any high level structures. This can lead to undesirable effects. For example when a song that consist out of alternating sections (e.g. chorus and verse) is shortened, it is very likely that a jump from one to another chorus or from a verse to another verse is used, due to their high similarity. If such a jump is used all the segments in between vanish, which changes the structure of the song. Figure 5.3 shows an example of this effect. Likewise when a song is enlarged, undesirable repetitions of segments can occur as shown in Figure 5.4. In this Section we will extend the algorithm to overcome this limitation.

The modeling of high-level constrains is based on the idea of *motion graphs* [KGP02]. Motion graphs have been developed in the context of computer animation. Given a corpus of motion capturing data with different labeled categories (like walking, running, jumping, etc) the goal is to synthesize new motion, that satisfies a certain sequence of labeled motions (e.g. first running for 10 meters, then jumping). The animation sequences are analyzed for similarities to create a graph of jump positions within data of the same label, to create generators for each label and between sequences of different labels to find transition candidates.

In our context the corpus corresponds to the input song and the labels correspond to the clusters extracted in Section 4.2; e.g. chorus, verse, etc. The main difference from the original animation use case is that we enforce duration constraints instead of spacial constraints and we do not use interpolation to create transitions, which would lead to low audio quality.

With this extension a wide range of new challenging applications are possible. The user can not only define the new total length and start and end positions of the song, but also the new sequence of labels and their individual duration.

The implementation is done by creating an adjacency matrix D_x for each label x. The graph to synthesize music of label x only contains edges that direct to bixels of label x and edges between different bixels with label x, which means that a label can be entered, shortened and enlarged but not left. To set up D_x we start with the full matrix D and remove all the undesired edges (edges that start with a bixel of label x and point to a bixel with a different label) in the graph by resetting their value to 1. A complete set of graph matrices for all labels of the song "Flood" by Tenpenny Joke is shown in Figure 5.5.

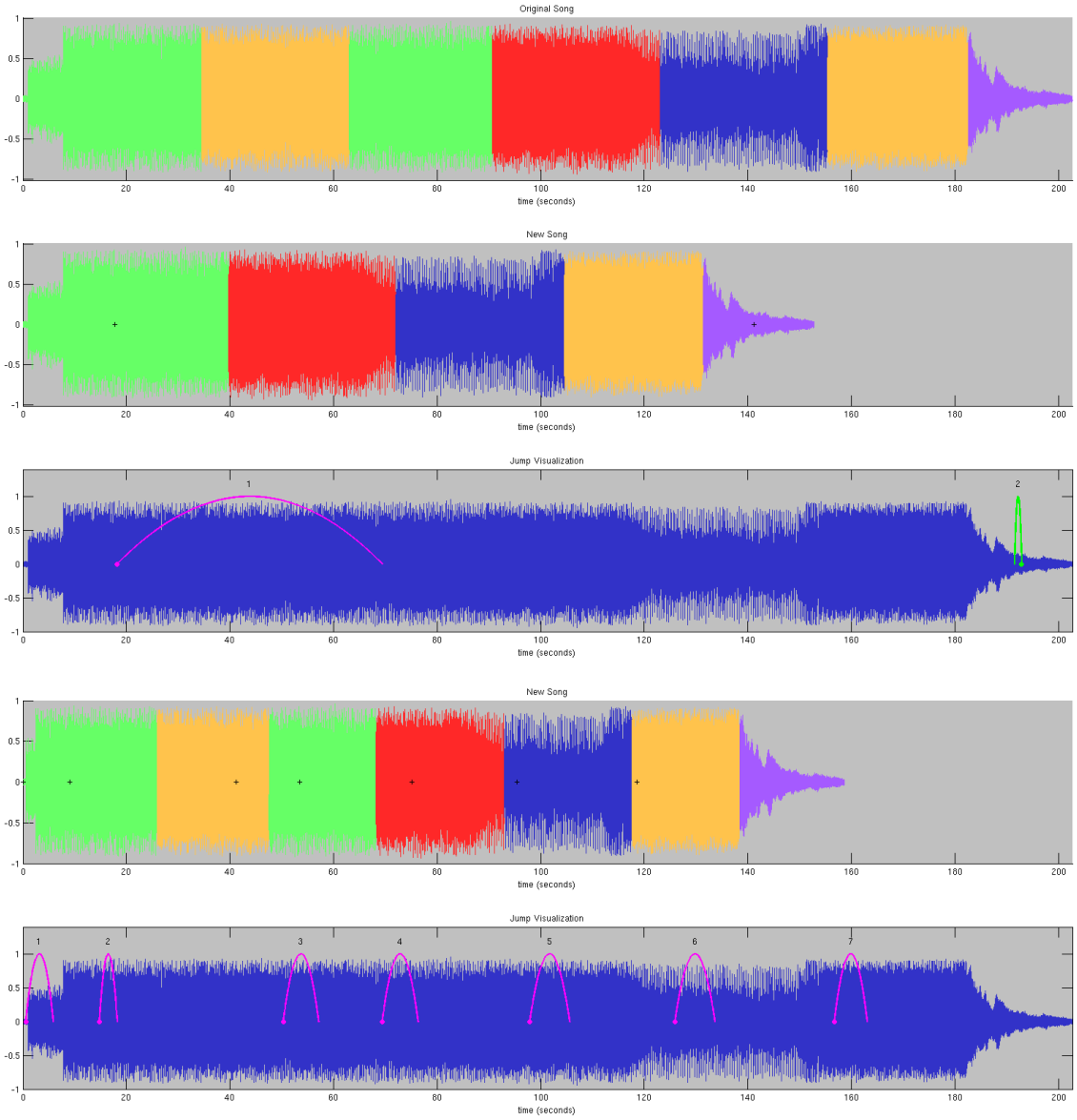


Figure 5.3: *Retargeted version of the song "Flood" by Tenpenny Joke to 75% of the original size with and without structural constraints. Plot description from top to bottom:*
(1) original song with segmentation.
(2) scaled song without structural constraints. The music structure is destroyed because the first yellow segment is lost, which is caused by the jump number one from the first to the second green segment.
(3) jumps of the scaled song without structural constraints.
(4) scaled song with structural constraints. The segmentation is preserved.
(5) jumps of the scaled song with structural constraints.

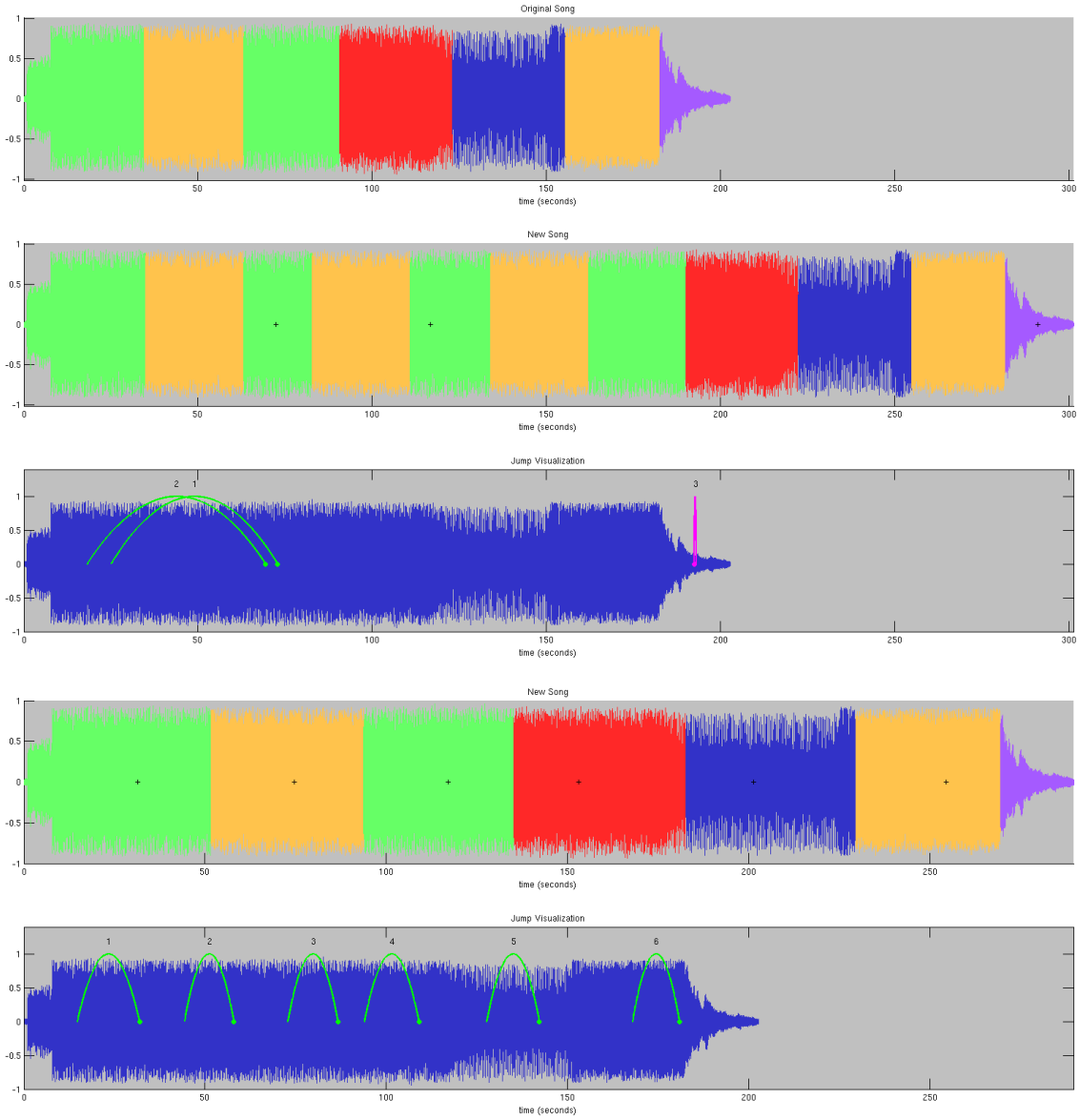


Figure 5.4: *Retargeted version of the song "Flood" by Tenpenny Joke to 150% of the original size with and without structural constraints. Plot description from top to bottom:*
(1) original song with segmentation.
(2) scaled song without structural constraints. The music structure is destroyed because green and yellow segments are repeated. The repetition is caused by the two backward jumps from the second to the first green segment.
(3) jumps of the scaled song without structural constraints.
(4) scaled song with structural constraints. The segmentation is preserved.
(5) jumps of the scaled song with structural constraints.

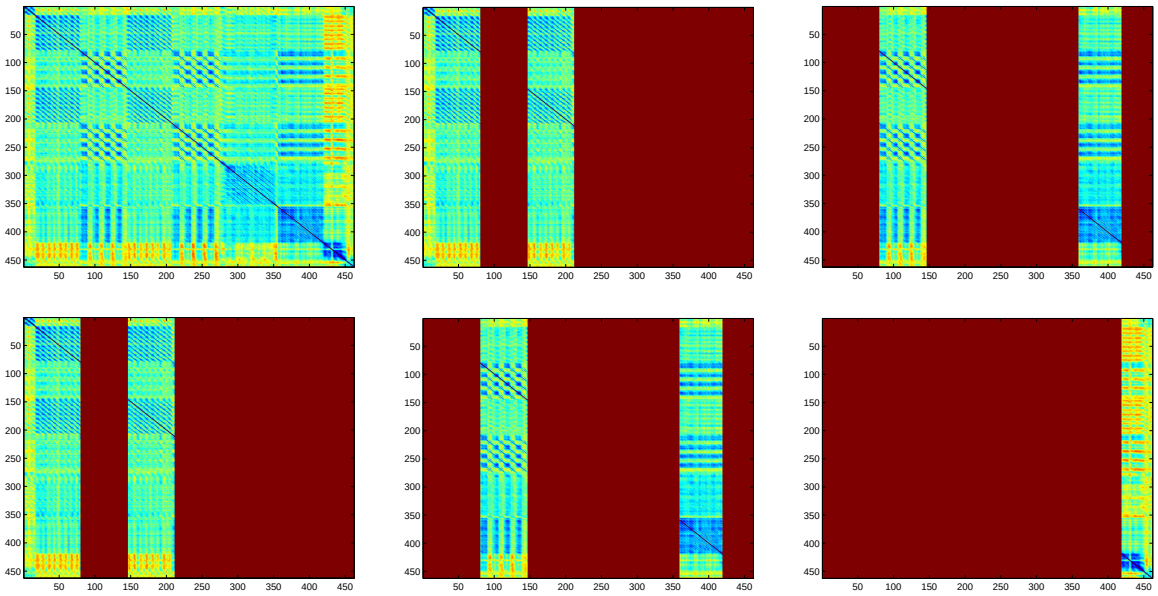


Figure 5.5: *Adjacency matrices for the song "Flood" by Tenpenny Joke. The first image shows the unmodified matrix of the full song. The other images are derived from this matrix and contain the adjacency for all the labels (1 to 5). See Figure 5.3 for an image of the song's segmented audio signal for comparison.*

Our dynamic programming method from Section 5.3 is modified as follows. The unified cost matrix in equation (5.4) becomes a stack of cost matrices one for each label x:

$$W(l_i, l_j)_x = \lambda I(l_j) + (1 - \lambda) D(l_i, l_j)_x \tag{5.6}$$

The recursive cost function (5.5) is modified by the cost matrix W_x depending on the function $X(k)$ which assigns a label to the bixel at position k in the output song:

$$C_s(i, k) = \min_{j \in \{1..N\}} \Big(C_s(j, k-1) + W(l_j, l_i)_{X(k)} \Big) \tag{5.7}$$

The overall procedure remains unchanged. The final output of our method is a path along bixels of the input song that defines the new output song with the desired new structure. The individual graphs not only generate content of a certain label, but they also enable us to find the globally optimal transitions between all the desired segments, which allows us to perform structural reshuffling operations.

The concept of editing the similarity matrix, or equivalently the adjacency matrix, is a very powerful concept. It can be used to model other kinds of constraints. For example it is simple to completely remove certain sequences from the synthesis process by removing all edges to these bixels. To protect parts of a song from modification all edges with the exception of the naturally following one have to be removed. Protection is a handy concept to prevent jumps that cut through vocal parts and destroy the meaning of the lyrics.

5.5 Post-processing

In the last step of the synthesis pipeline the abstract path representation from the previous step is transformed into the final audio signal, a sequence of samples. The path defines how the bixels in the original song have to be rearranged to create the output song. Bixels at jump positions might need some processing to make sure that no distortions are created by sudden steps in the waveform, audible as a short "click" sound. Two different solutions for this problem have been implemented, blending and alignment. Depending on the properties of the signal the user can choose the desired method. Blending works well for most songs, but experiments have shown that it is outperformed by alignment on signals with high frequencies and amplitude where blending tends to introduce damping. Blending mixes the adjacent bixels inside an overlapping window of 0.2 seconds using a linear mixing weight function. The alignment strategy searches for a small offset between the two bixels so that the waveforms have optimal continuity. The search is performed inside a neighborhood of 0.02 seconds using the sum of squared differences in a window of 32 samples as error measure. The alignment method shortens the song by a small amount, typically about 100 samples (~0,002 seconds) per jump position, which is negligible for most use cases.

The results created by the path search algorithm can achieve a duration precision up to the length of a beat. For some applications, like the alignment of audio events to video frames as presented in Section 6.6, a precision up to single samples is required. Given our dynamic programming based result only an additional scaling in the range of 0 to 3 percent is required to achieve this. For such small scaling factors a timescale-pitch method is well suited. In our pipeline we use an implementation[1] of the WSOLA timescale-pitch method by *Verhelst et al.* [VR93].

Although the analysis is performed on a mono signal any number of audio channels can be synthesized. A multi-channel input is converted into mono by computing the average of all channel signals. The synthesis returns the result as a path, which then can be applied to every channel of the unmodified input individually to create a multi-channel result.

[1] http://tarsos.0110.be/artikels/lees/Audio_Time_Stretching_-_Implementation_in_Pure_Java_Using_WSOLA

6

Results

In this chapter we present the interactive user interface that has been developed to control the synthesis process (Section 6.1) and different experimental results obtained with our method. We explored diverse applications such as: content aware retargeting (Section 6.2), music loops (Section 6.3), singing removal (Section 6.4), music reshuffling (Section 6.5) and video editing (Section 6.6). All results were obtained on an Intel Core2 Quad Q6600 CPU with 8 GB of main memory. For an input audio file of 3 minutes ($N = 300$), the feature extraction and segmentation takes about 1 minute (MATLAB implementation). The runtime of the synthesis process, mostly implemented in C++, depends on the output duration. To enlarge the music by a factor of two (6 minutes, $M = 600$), it takes about 160 milliseconds. To reduce the input song to 1 minutes ($M = 100$) about 30 milliseconds are needed. Therefore our method allows for real-time synthesis and user interactivity.

All audio sequences and video results presented in this chapter are contained in the supplemental materials.

6.1 Interactive software framework

A graphical user interface has been created where the user can interactively control and evaluate the synthesis process. The analysis step, namely computing the beat-aligned self-similarity matrix and the segment boundaries and labels, are performed as a pre-processing step and get stored in memory. After the pre-processing, the synthesis step can be performed in real-time, which allows user interaction.

The input file formats MP3 and WAV are supported. Unfortunately it is not straight forward to supply a new input song to the pipeline. Beat tracking has to be performed by an external

application, e.g. BeatRoot[1], and has to be stored in a CSV file. The file has to contain an increasing sequence of beat positions in seconds separated by commas. Missing beat positions at the beginning or at the end of the song will be extrapolated. The music and beat file have to be named the same. In addition the file name has to start with a unique number, which is used to reference it.

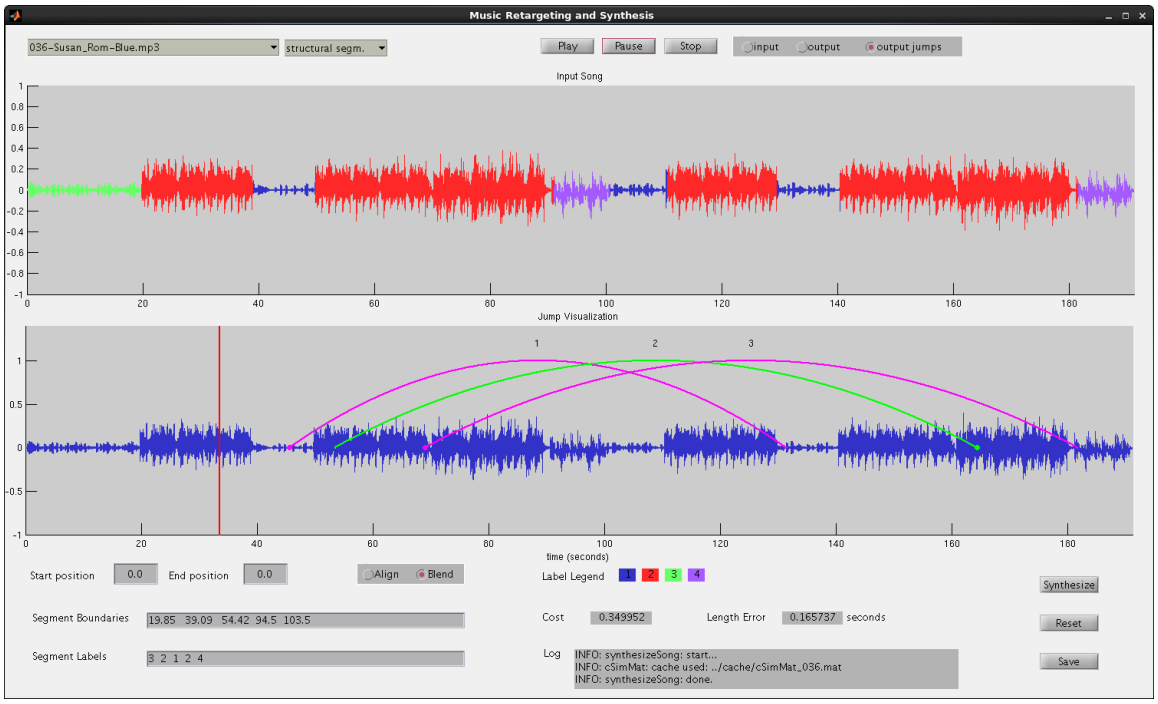


Figure 6.1: *Screenshot of the interactive graphical user interface. The plots show the original and a reshuffled version of the song "Blue" by Susan Rom.*

Figure 6.1 shows a screenshot of the interface. The input music and its segmentation can be selected in the selection boxes in the upper left corner of the interface. The topmost plot shows the input music with its segmentation, if any. The second plot displays the output, either the final signal with segmentation or by visualizing the jumps in the input signal. Both signals can be played using the internal audio player by selecting the corresponding radio button in the first row. The player can seek to a certain position in the song by clicking into the plot of the currently active signal.

The synthesis constraints can be specified in the lower part of the interface. The user can define the start and the end positions (in seconds) in the input song, the new positions of the segment boundaries and their new labels. Further the post-processing can be switched between alignment and blending. A click on the "Synthesize" button starts the synthesis process. Warnings and status messages are displayed in the log. As soon as the synthesis process is finished, the cost and the error in length are displayed. Pressing the "Reset" button restores the segment boundaries and labels for the output to the one of the input. "Save" stores the output signal to disk.

[1] http://www.eecs.qmul.ac.uk/~simond/beatroot/

6.2 Content aware retargeting

A core use case of our algorithm is the content aware retargeting of music to a new output duration. The new song starts and ends with the same tune as the input, the high scale structure is preserved and the duration is enlarged or shortened. Figure 6.2 shows a shortening to 75% and Figure 6.3 an enlargement to 150% of the song “Flood” by Tenpenny Joke. The last segment was excluded from retargeting. Because of its progressive amplitude change it is impossible to find any high quality jump position.

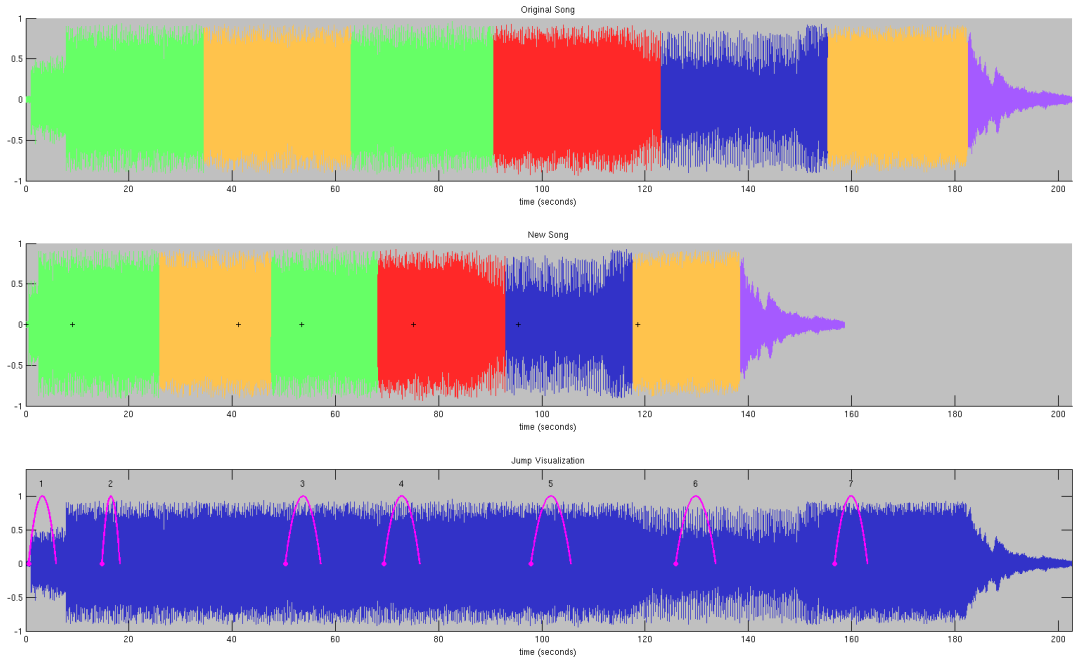


Figure 6.2: *Retargeting of the song “Flood” by Tenpenny Joke. Top: original song. Middle: retargeted version with 75% of the original size. Bottom: the associated jumps.*

6.3 Infinite music

An interesting application of our algorithm is the creation of music of infinite length. This is for example needed for interactive games, like Zelda or Tetris, where the length of a certain scene or action is unknown. One way to create such music is to create a song, that can be played in a loop. This can be easily achieved with our method, by using good start and end positions for the path search. Such positions can be defined by the user or they can be extracted automatically. Local minima in the beat aligned similarity matrix are good candidates. Figure 6.4 shows an example of an infinite guitar song with 60 seconds duration. The start and the end position was set to the 60 second time stamp. Additional high-quality loop songs can be found in the supplemental material. With the assistance of our method it is also possible to generate outro sequences from any position in the loop to the end of the song on the fly, e.g. when the current level or scene of the game is ending.

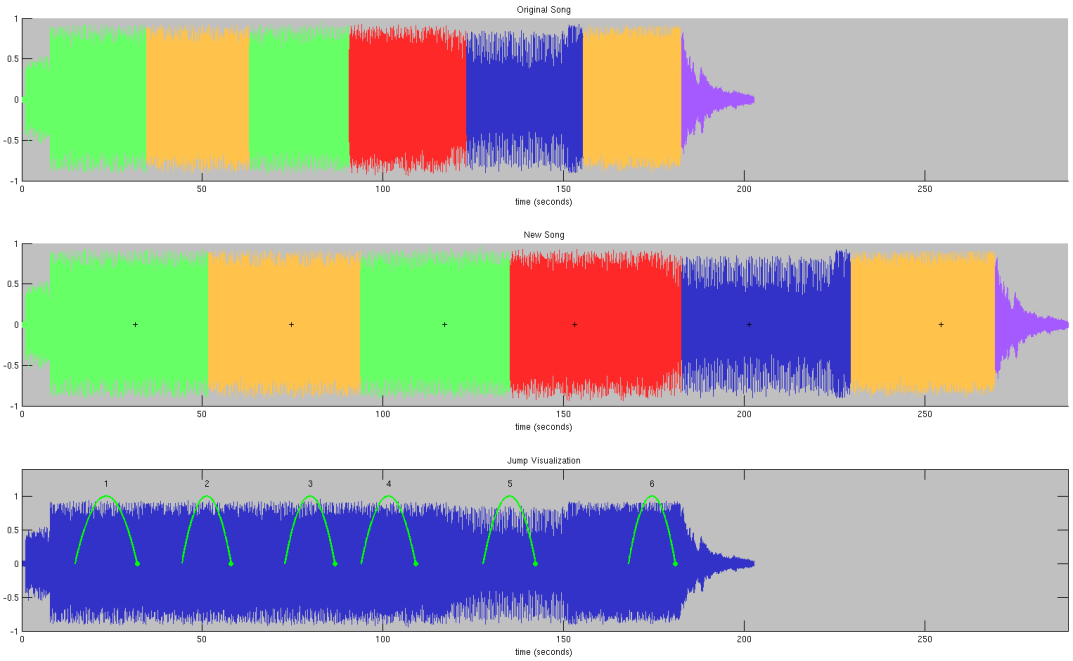


Figure 6.3: *Retargeting of the song "Flood" by Tenpenny Joke. Top: original song. Middle: retargeted version with 150% of the original size. Bottom: the associated jumps.*

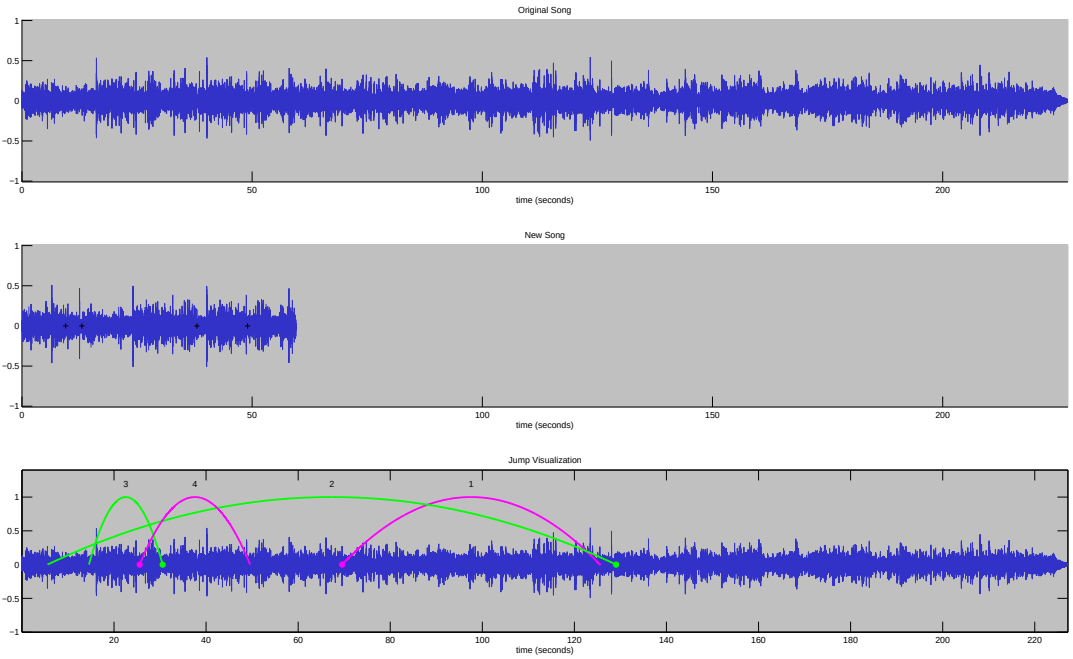


Figure 6.4: *Infinite music created from the song "Rumba Alemana" by El Perez. Top: original song. Middle: the retargeted music by the proposed algorithm to 60 seconds duration, that will be played in loop mode to obtain infinite music without audio discontinuities. Bottom: the associated jumps.*

6.4 Singing removal

Another use case is the removal of parts of the song and automatically restoring its continuity. Imagine that we don't like the lyrics of a song or we want an instrumental version that can be used as background music. Given a segmentation into vocal and non vocal parts our method can create a version that only consists out of instrumental parts. Vocal parts can be extracted automatically using speech detection [JR07] or from lyric (LRC) files[2]. The LRC file type is no formal standard, but it is very popular for karaoke applications and various online sources[3] provide such files for most mainstream songs. The files are plain text and on each line the start time along with the corresponding text is defined. Instrumental sections are encoded with time stamps without any lyric text. Figure 6.6 shows an excerpt of the LRC file of the song "Trouble" by the band Coldplay and Figure 6.7 the resulting segmentation and synthesis without singing.

We manually labeled the vocal parts in the Russian song "Chuchelo" by Nastya Yasnaya and synthesized an instrumental version of the song. The segmentation, the resulting song and the jumps can be seen in Figure 6.5. The audio file of this result is part of the supplemental material.

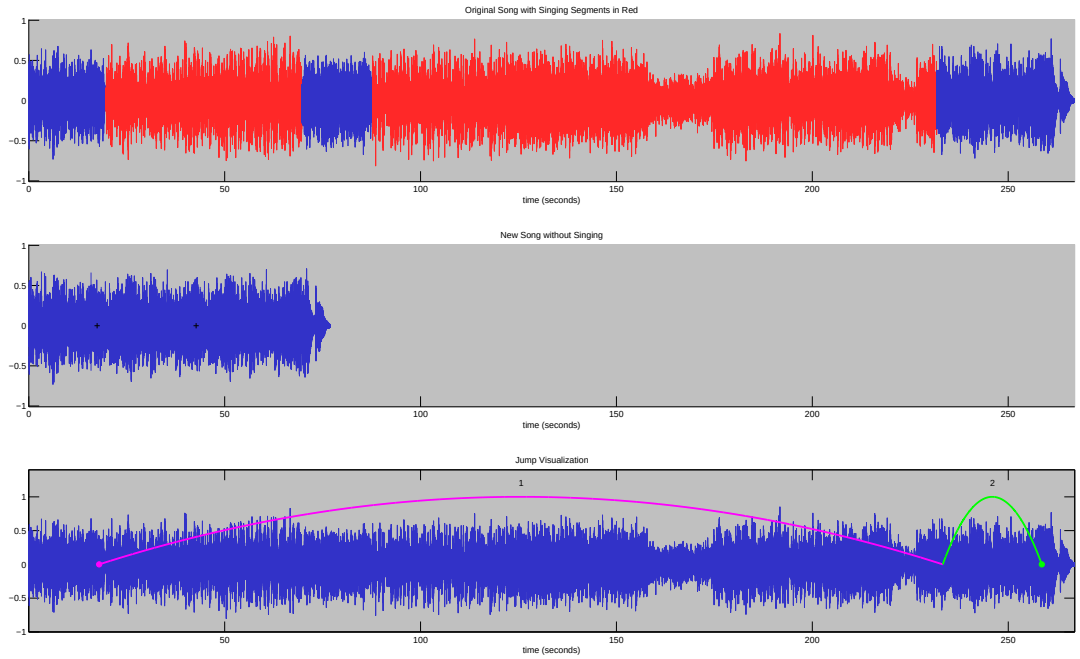


Figure 6.5: *Singing part removal for the Russian song "Chuchelo" by Nastya Yasnaya. Top: original song with singing segments in red. Middle: the retargeted music by the proposed algorithm. Bottom: the associated jumps.*

[2]http://en.wikipedia.org/wiki/LRC_(file_format)
[3]http://www.lyrdb.com/karaoke/

```
...
[01:12.000]A spider web, and I'm caught in the middle
[01:17.000]Oh I turned to run
[01:20.000]
[01:23.000]The thought of all the stupid things I've done
[01:27.000]
[01:30.000]And oh, I never meant to cause you trouble
[01:34.000]
[01:36.000]And oh, and I never meant to do you wrong
...
```

Figure 6.6: *Excerpt of the LRC file of the song "Trouble" by the band Coldplay. Each line defines the beginning of a time slot with its start time and the corresponding lyrics.*

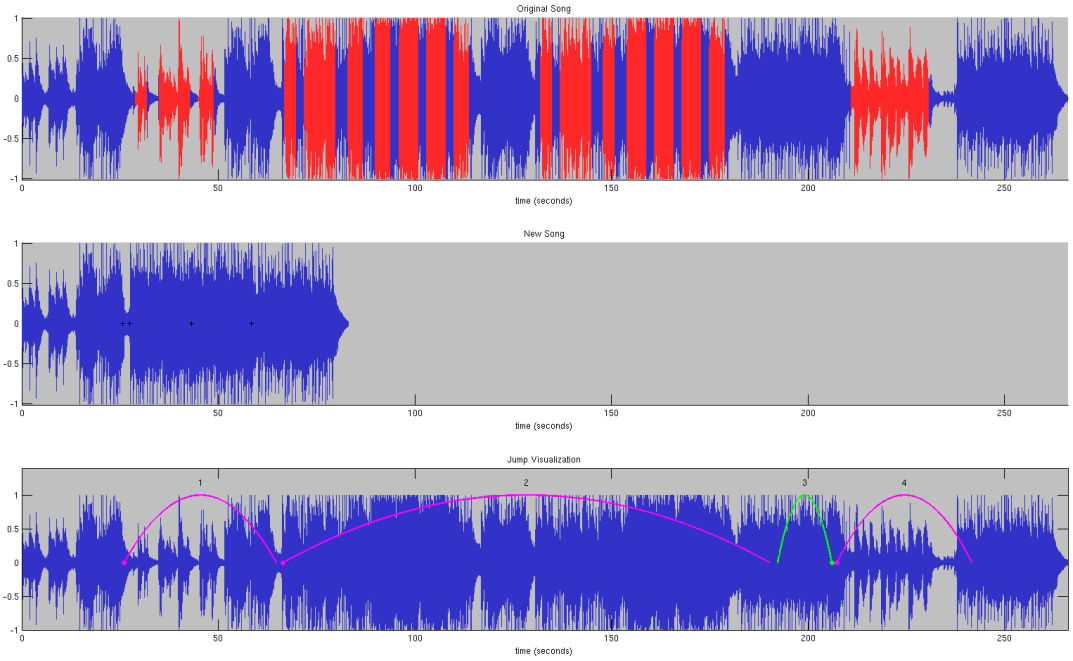


Figure 6.7: *Singing segmentation and synthesized version without singing of the song "Trouble" by the band Coldplay. Top: original song with singing segments in red. Middle: the retargeted music by the proposed algorithm. Bottom: the associated jumps.*

6.5 Reshuffling

The availability of the structural segmentation and the possibility to synthesize each label individually enables many powerful applications. We can create songs from only one label, e.g. only the chorus parts, to create a long pure refrain song. A more powerful operation is the reshuffling of the original structure into a completely new structure. Figure 6.8 shows an example where we create a summary of the high-level structure of a song. The new music contains only half as many red and violet parts and only a third of the blue parts, but each segment type appears at least once. This synthesis result could serve as a music thumbnail for efficient

browsing. Figure 6.9 shows an example of a reordering of the song structure. The green label is moved between two red labels, which forms a structure that does not exists in the original song. Assisted editing operations of this nature offer an efficient way to remix and transform existing music works into new creations.

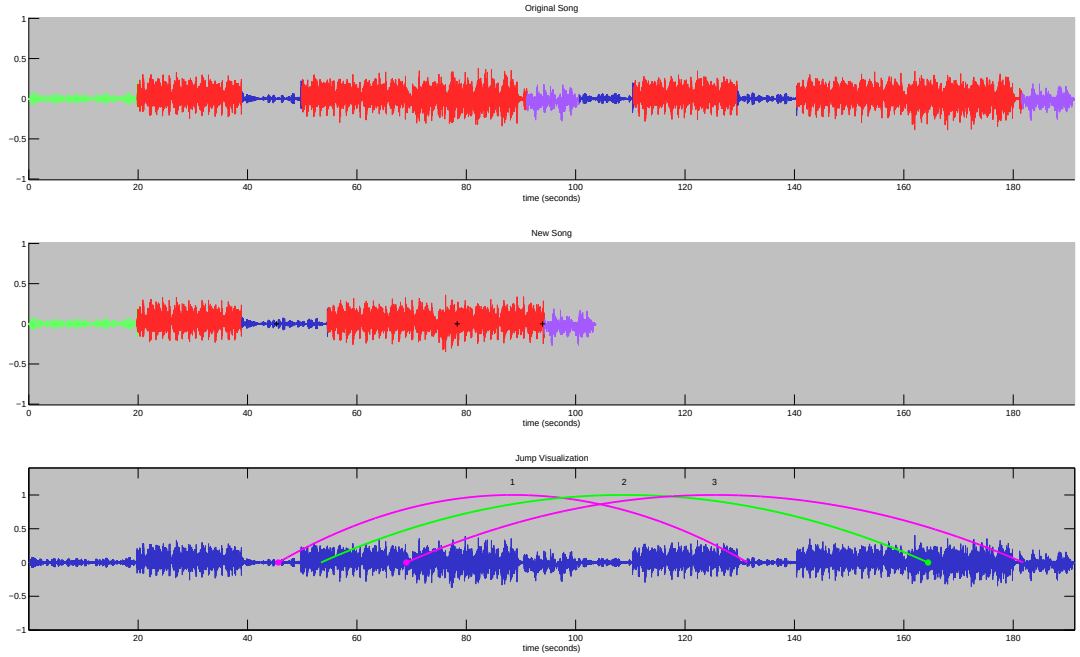


Figure 6.8: *A reshuffling of the song "Blue" by Susan Rom. Top: original song with the original segmentation. Middle: the reshuffled song with the new summarized structure. Bottom: the associated jumps.*

6.6 Video editing

The proposed method can be used as an automatic assistance for video editors, who have to post-process existing video footage with background music. The music is often composed to match the content of the video. The removal or addition of video scenes requires manual processing of the audio or a completely new recording of the music. Using our technique it is possible to restore the continuity by retargeting the music to the new duration. In addition it is possible to enforce time frames where the video and the audio have to be synchronized, for example for dialogues, environment sounds or special effects.

We show three different results where scenes from a movie are removed and the audio is re-synthesized using our method: Manual scene removal with and without constraint, and automatic video conversion (cf. Figure 6.10). We compare each of them to the simple manual approach where the audio is cut at the same position as the video and the resulting borders between audio slices are blended. We call this the "cut-and-blend" approach. See Figure 6.11 for an illustration of this technique.

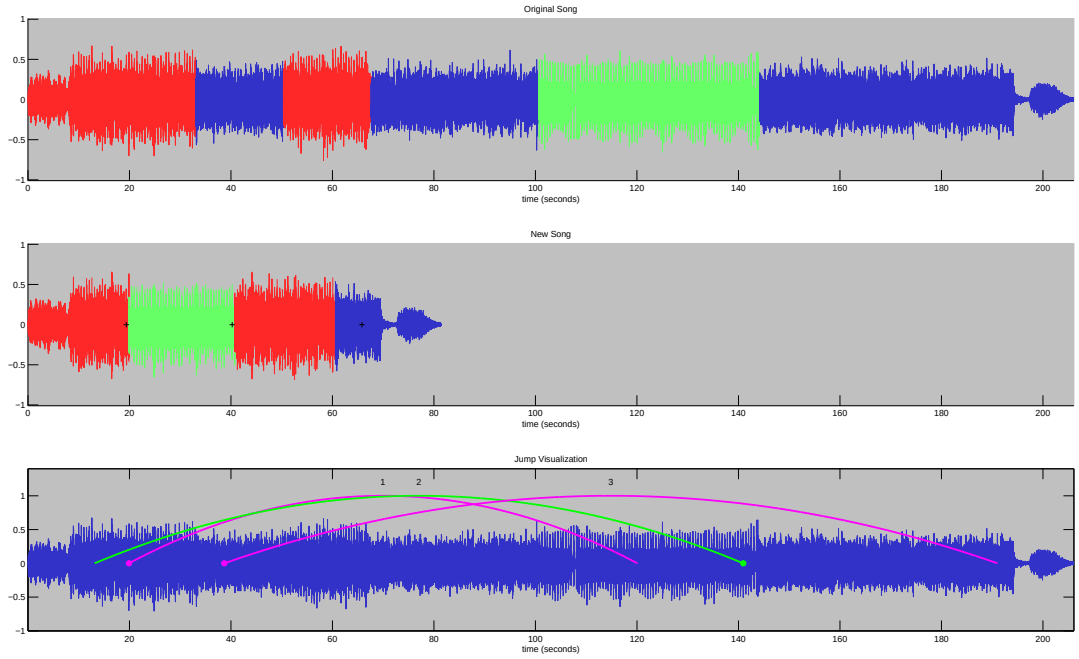
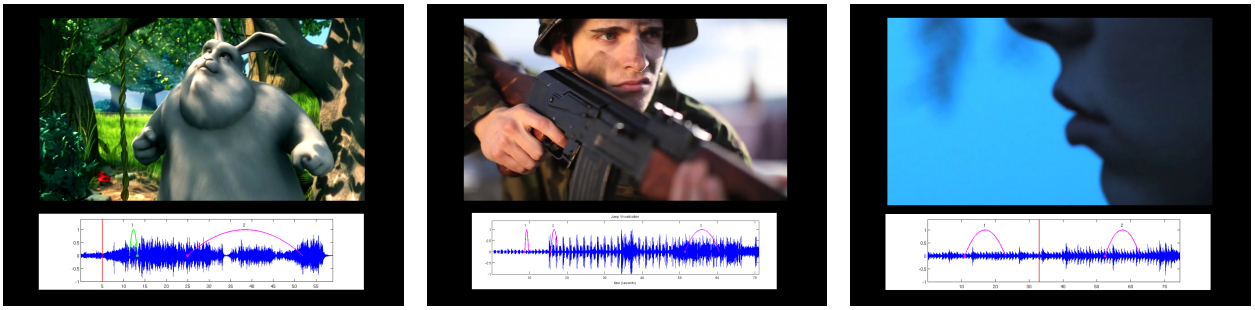


Figure 6.9: *A reshuffling of the song "Victory" by Alexander Blu. Top: original song with the original segmentation. Middle: the reshuffled song with the new structure. Bottom: the associated jumps.*



(a) (b) (c)

Figure 6.10: *Video editing results screenshots with a representative video frame on top and the waveform with the associated jumps on the bottom:* (a) *Big Buck Bunny.* (b) *Better World.* (c) *Oceania.*

6.6.1 Assisted video editing

Imagine a video editor wants to change the action level of the selected sequence of Big Buck Bunny (directed by Sacha Goedegebure, 2008). Please refer to the supplementary material for the original and both modified version of the video sequence. The scene containing the rope jumping and the part were the rodents walk over the field get removed, including the corresponding audio parts. The audio and video duration is reduced from 60 to 33 seconds. After the scene removal, the standard approach to process the audio, would be to blend the music at the cut positions (cut-and-blend approach). With our method it is possible to retarget

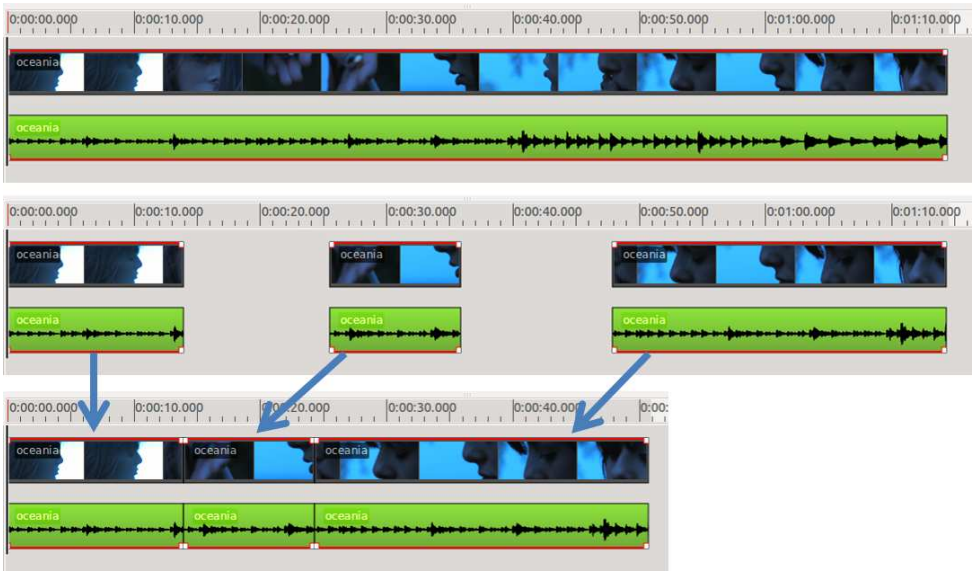


Figure 6.11: *Traditional process in video editing: given the original video and the associated audio track (top), the editor can cut some scenes (middle) and finally, the remaining ones, along with their associated audio, are stitched together and the audio is blended (bottom).*

the background music to the new duration of the movie, which leads to the intended result without manual work and without noticeable audio discontinuities. The input to our algorithm is the original audio track of the movie with a segmentation with two different labels; (1) for the parts that are contained in the edited version of the video and (2) for all the removed parts. The start and the end positions remain the same as in the input to ensure continuity to the adjacent scenes. The output music consists out of only one segment with label 1 of the desired duration.

6.6.2 Assisted video editing with constraints

If the audio track of a video contains non musical sound effects (e.g. an explosion), or the music is custom tailored to scene's events, it is important to re-align these audio effects with the associated video frames in the edited movie. This is possible with our method. We demonstrate this use case by editing the music video "Better World" by the band Saturday's Tinitus (directed by Reto Troxler, 2010). The video contains different scenes of a story and shots of the band playing. We want to remove all the scenes that show the band to get a compressed version of the story. See Figure 6.13 for a visualization of the removed scenes. One scene (after 34 seconds of playback) contains explosion effects that have to be aligned with the corresponding explosion sounds in the music. The constraint enforcement is modeled by a segmentation. The time window that contains the explosions forms a segment with a label and all the remaining parts form segments with a different label (see Figure 6.13). As in the previous example (Section 6.6.1) the start and the end position remain the same. The output segmentation has the same structure as the input but the segments without constraint have a new duration. The duration of these segments is reduced by the length of the scenes that have been removed inside of them. The whole video is resized from 71 to 60 seconds. The cut-and-blend version of this example has a perfect alignment of the explosions but very annoying discontinuities of the rhythm and

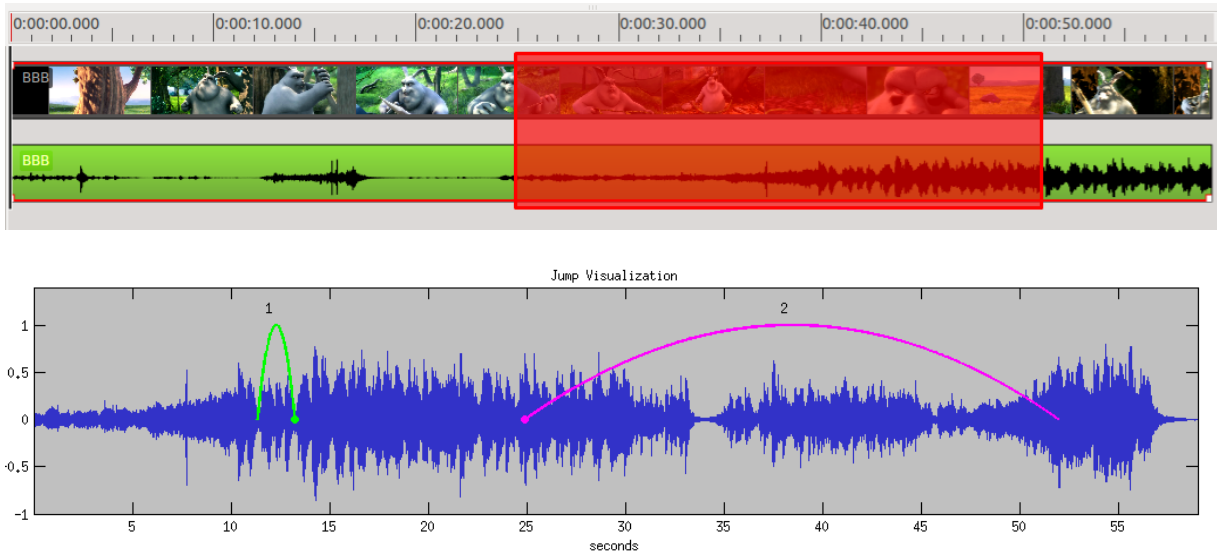


Figure 6.12: (1) *Video editor time line of the "Big Buck Bunny" video sequence with the scenes that are removed marked in red.* (2) *The resulting jumps to retarget the audio. The forward jump (purple) skips the removed part using an optimal transition and the backwards jump (green) enlarges the audio by a small amount to compensate for the transition.*

melody at the cut positions. Our method succeeds to align the explosions and creates high quality transitions.

6.6.3 "restricted" to "general audience" conversion

The Motion Picture Association of America's film-rating system rates a film's thematic and content suitability for certain audiences. At one end of the spectrum, G refers to general audience and all ages are admitted. At the other end, R stands for restricted: under 17 requires accompanying parent or adult guardian.

Our approach can be used to remove restricted scenes from a movie to transform "restricted" movies to "general audience". In the movie "Oceania" (directed by Harpreet Dehal, 2008) we manually selected the restricted sequences. The video sequence consists out of a scene of 74 seconds duration, where girls are standing on the shore and smoke. It was our goal to create a version of this sequence without any smoking scenes. The new video has a length of 52 seconds and consists out of a single input and output segment. The supplemental material contains the retargeted version and the one using the cut-and-blend technique for comparison. The cut-and-blend result has annoying breaks in the rhythm, while our method creates music with inaudible transitions.

To convert hundreds of restricted movies to "general audience", automatic classifiers for violence [NSGS11] or nudity [DPN08] could automatically detect the positions of the sequences containing restricted contents, and then our algorithm can be applied to retarget the associated music to the duration of the cut video.

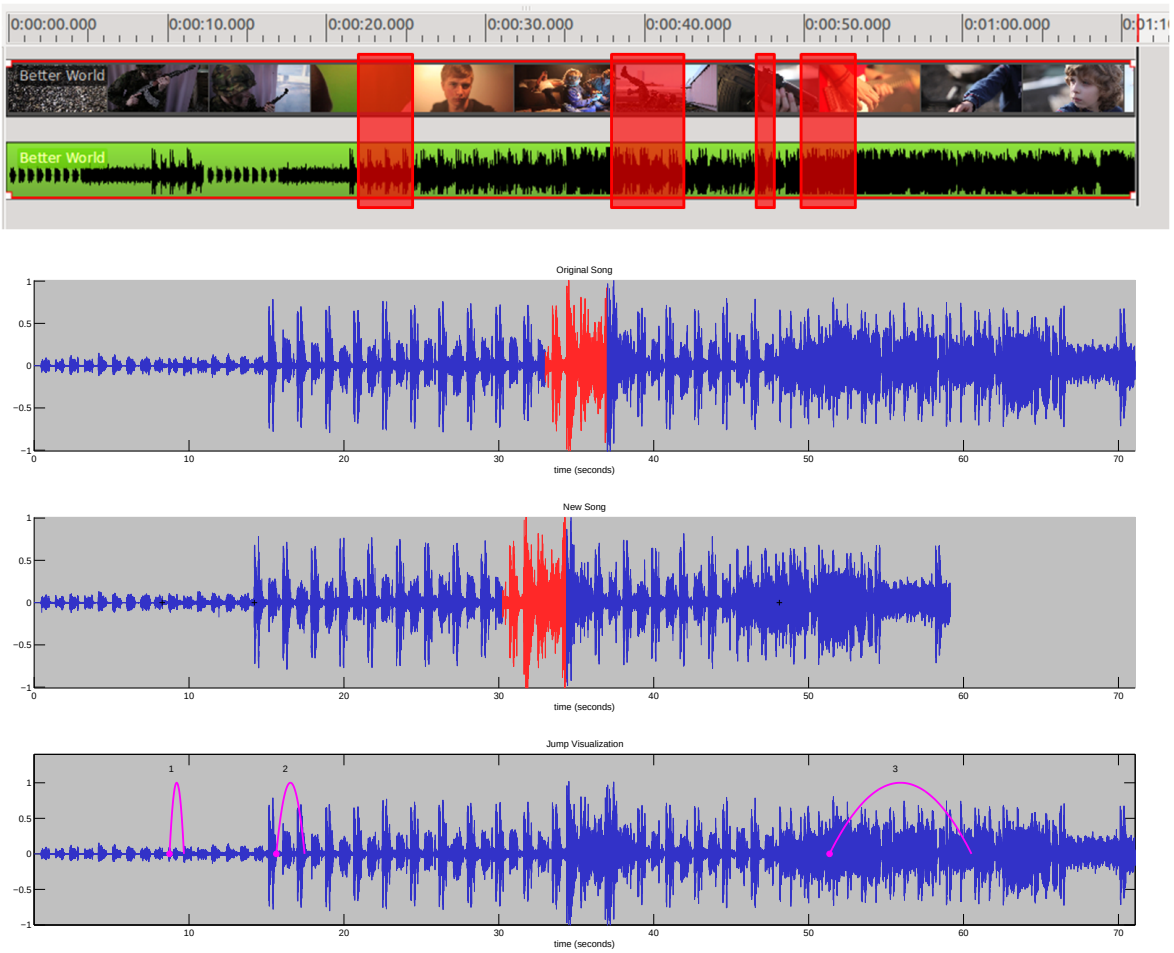


Figure 6.13: (1) *Video editor time line of the "Better World" video sequence with the scenes that are removed marked in red.* (2) *The first plot shows the input song with segmentation. The red segment has to be aligned to the original video frames. The second plot shows the final audio result and the third plot the associated jumps to retarget the audio.*

6.7 Scaling comparison

Table 6.1 contains the results of a comparison of our method to the existing retargeting applications *Echo Nest Earworm* and *Paul's Extreme Stretch* version 2.2. The input song was scaled with each method by the desired scaling factor. The table shows the duration error and the subjective perceived quality of all the results. The quality values range from *good* (unnoticeable), *sufficient* (slightly annoying) to *bad* (very annoying).

Earworm is able to create good results only for the song "Open Fields Blues" by the artist Hiiragi Fukuda, which is an example song provided by the developers of the software. Earworm is unable to find any solution for the song "Flood" by the artist Tenpenny Joke and returned the input song for the 0.75 case, hence the high duration error, and no song for the 1.5 scaling case. This is probably caused by the inability to extract any meaningful feature descriptors from the audio signal. The song "Rumba Alemana" crashes the application with a segmentation fault independent of the desired length.

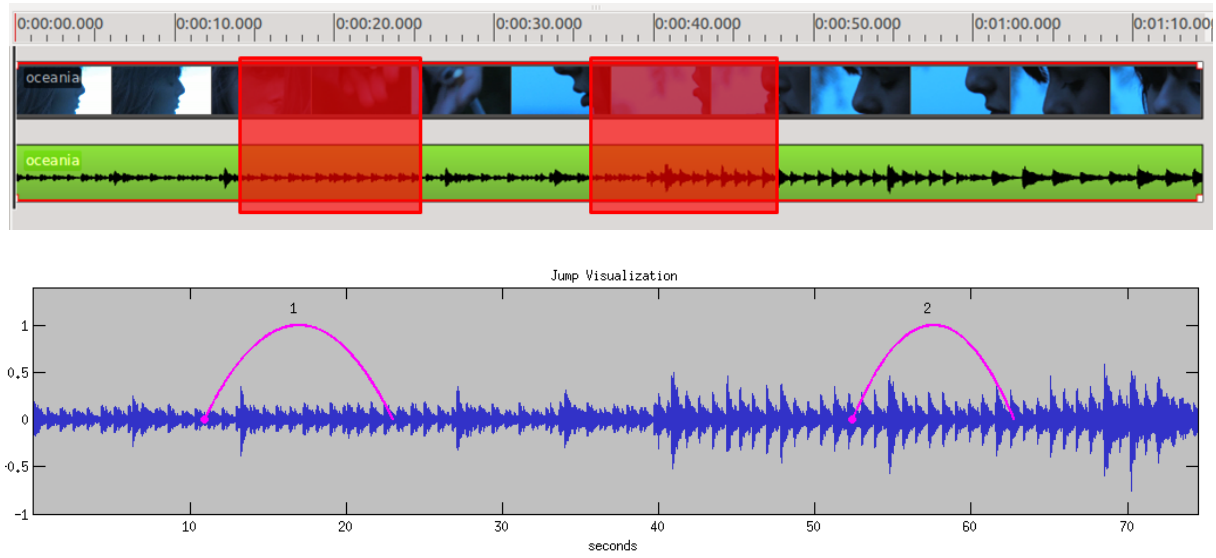


Figure 6.14: (1) *Video editor time line of the "Oceania" video sequence with the scenes that are removed marked in red.* (2) *The resulting jumps to retarget the audio.*

Paul's Extreme Sound Stretch creates solutions that are very close to the desired length, but the quality is very poor. The method introduces strong echos and distortions.

Our method could create solutions up to the precision of the beat length (about 0.5 seconds). As Earworm's algorithm optimizes for quality and not for duration precision, we also made a trade-off between both goals. For this comparison our method searches for the best solution within a tolerance of 5 seconds. With the exception of the shortening of the song "Open Fields Blues", our method is always able to produce good results.

Hiiragi Fukuda - Open Fields Blues (161.51 sec.)						
	Earworm		*Paul's Method*		*Our Method*	
scaling factor	error (sec.)	quality	error (sec.)	quality	error (sec.)	quality
0.75	+6.16	good	-0.13	bad	-4.50	sufficient
1.5	+5.82	good	-0.27	bad	-4.75	good
Tenpenny Joke - Flood (212.77 sec.)						
0.75	+53.19	good	-0.58	bad	-2.49	good
1.5	n/a[1]	n/a[1]	-1.16	bad	-4.50	good
El Perez - Rumba Alemana (227.96 sec.)						
0.75	n/a[2]	n/a[2]	+0.12	bad	-1.75	good
1.5	n/a[2]	n/a[2]	-0.44	bad	-0.50	good

Table 6.1: *Comparison of Echo Nest Earworm and Paul's Extreme Stretch to our method.*

[1] Fails with the message *MEMORY ERROR*
[2] Application (Python) crashes with a segmentation fault

7

Conclusion and Outlook

7.1 Discussion

In this thesis we presented a novel method for music retargeting and editing inspired by computer graphics and animation. It enables content aware resizing and reshuffling, using a combination of beat-aligned self-similarity matrices and segmentation for structure preservation. The synthesis space is modeled as a directed graph and the globally optimal solution is found by searching the k-stops shortest path using dynamic programming.

We showed the usefulness of our method by presenting high quality results for a wide range of challenging applications, such as duration changes, infinite loops, content removal, reshuffling and video editing. These applications would be difficult or impossible to achieve with previous approaches. There are still many additional applications that could be explored, for example the creation of high quality music summaries for efficient browsing of collections or the automatic creation of seamless music playlists with optimal ordering and transitions with minimal distortion. The retargeting process could also be guided by additional high level feature sources like the motion of a dancer [SNI06], robot kinematics or the content of a video. This would allow to synthesize music, which is automatically synchronized to key events in animation or video data.

Our solution is built on ideas from computer graphics research and demonstrates, that audio processing may benefit from their concepts. We think that many other techniques recently developed in image and video processing might have the potential to inspire novel approaches in audio processing, and vice versa.

7.2 Limitations

For most music genres the proposed method performs well, with the exception of audio data with no or hardly any self-similar parts. In these cases a solution is still returned, but the resulting transitions are very annoying for the listener. Examples that fall into this category are certain rock and some classic music, whose structure is purely progressive in nature. Over time the loudness, melody or rhythm changes incrementally. The song "Bohemian Rhapsody" by the band Queen[1] is an extreme example. Potential jump positions are only inside a small neighborhood of a beat and if used they introduce discontinuities.

Other issues are caused by vocals. If the vocal track is not very pronounced, it has the same influence in the descriptor as any other instrument. It can happen that jumps are used from inside a vocal part to an instrumental part, because all the instruments and the melody matches perfectly. While acceptable for instruments, the human perception is very sensitive to speech. The resulting word fragments are noticeable. This limitation could be solved by adding voice detection [JR07] as an additional component to the descriptor.

Our method can protect and remove parts of the input music, but with our current energy minimization strategy it is not possible to enforce the usage of a part without specifying a hard positioning constraint. The importance function is only a guidance function. It increases the probability that a certain part is contained in the solution, but there is no guarantee that it is used under all circumstances.

The proposed method does not contain any explicit repetition control. The enlargement of a song could lead to a solution, where a small slice is repeated over and over until the requested length is reached. This is not desirable. But experiments did show that this is not an issue in practice. The resulting k-stops shortest paths tend to contain only a low number of jumps and thus, the number of repetitions is also small.

7.3 Future work

The feature descriptor could be improved by incorporating information about vocals, as lyrics cause the most issues in practice. Such coefficients could be based on a singing detection algorithm [JR07] or karaoke data. A different approach to improve the robustness of the similarity measure could be to combine timbre and chroma features into one similarity matrix.

The extracted song model contains the high level structure (chorus, verse, bridge, etc.) and very low level structure (beats, onsets), but it lacks knowledge about medium sized structures, e.g. melody. A hierarchical clustering of the existing beat descriptors or an additional melody description [Mar06] could provide this data.

The quality of the jump positions created by the proposed method is good, despite the simple strategy to jump directly from the end to the beginning of beats, the onset position. For bixels where the onset is not very pronounced, an optimized jump position could increase the jump quality. Within beat optimization could be performed by time warping both bixels to the same

[1] http://www.youtube.com/watch?v=irp8CNj9qBI

length and finding the most similar regions using cross correlation.

The proposed method has a lot of further potential. For example the pipeline could be extended to perform combined temporal retargeting of video and audio with unified constraints and optimization. Such a method could assist or automate a variety of additional video editing tasks.

A

Appendix

A.1 Glossary

bixel	description of samples that belong to a beat, a contraction of the words beat and pixel
CSV	comma separated values
FFT	fast Fourier transform
HMM	hidden Markov model
Mel	Mel scale, a perceptual frequency scale
MFCC	Mel frequency cepstral coefficients
jump	transition (edge in the graph) between two nodes that do not naturally follow in the input music
OLA	overlap add method
PSOLA	pitch synchronous overlap add method
SIFT	Scale-invariant feature transform
STFT	short-time Fourier transform
WSOLA	waveform similarity overlap add method

A.2 Software used

MATLAB Programming environment. *http://www.mathworks.ch/products/matlab/*

BeatRoot Beat extraction. *http://www.eecs.qmul.ac.uk/ simond/beatroot/*

MFCC We used a modified version of the MATLAB implementation by Kannu Metha.
http://www.mathworks.com/matlabcentral/fileexchange/23119-mfcc

Spectral Clustering Normalized cuts segmentation for MATLAB.
http://www.cis.upenn.edu/~jshi/software/

Pitivi Video editor. *http://www.pitivi.org/*

FFmpeg Batch audio and video processing. *http://www.ffmpeg.org/*

WSOLA Audio timescale-pitch method in Java.
http://tarsos.0110.be/artikels/lees/Audio_Time_Stretching_-_Implementation_in_Pure_Java_Using_WSOLA

Chroma Toolbox for MATLAB.
http://www.mpi-inf.mpg.de/resources/MIR/chromatoolbox/

Temporam Toolbox for MATLAB.
http://www.mpi-inf.mpg.de/resources/MIR/tempogramtoolbox/

A.3 Parameters

Analysis		
Parameter Name	*Value*	*Description*
Audio sample rate	44,100 Hz	Input and output audio sample rate
Audio bit rate	16	Number bits per sample
Silence window	0.1 sec.	Silence detection window size
Silence threshold	0.01	Silence detection threshold for no silence
MFCC bin count	40	Number of MFCCs
MFCC window size	beat size	Provided by beat tracking
Dynamics filter size	5	Size for the filter to add temporal consistency
Noise filter size	5x5	Gaussian smoothing filter
Novelty kernel size	96	Size of the novelty detection kernel
Novelty threshold	0.1	Novelty peak threshold
Cluster count	1–5	Number of structural elements (labels)
Synthesis		
Parameter Name	*Value*	*Description*
λ	0.5	Weighting of data vs. smoothness term
$I(l_i)$	0	Inverse importance function, all bixels are equally important
Blending window	0.2 sec.	Duration of the blending window
Alignment window	0.01 sec.	Duration of the alignment window
Alignment error window	32 samples	Number of samples used for error computation

Table A.1: *Default pipeline parameters used for all results if not explicitly defined.*

Bibliography

[AS07] Shai Avidan and Ariel Shamir. Seam carving for content-aware image resizing. *SIGGRAPH*, 26(3), 2007.

[Ash01] M. Ashikhmin. Synthesizing natural textures. In *ACM Symposium on Interactive 3D Graphics*, pages 217–226, 2001.

[BP92] Judith Brown and Miller S. Puckette. An efficient algorithm for the calculation of a constant q transform. *Journal of the Acoustical Society of America*, 92(5):2698–2701, 1992.

[BVZ01] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23:1222–1239, 2001.

[BW05] Mark A. Bartsch and Gregory H. Wakefield. Audio thumbnailing of popular music using chroma-based representations. *IEEE Transactions on Multimedia*, 7(1):96–104, 2005.

[CBBJR03] M. Cardle, S. Brooks, Z. Bar-Joseph, and P. Robinson. Sound-by-numbers: motion-driven sound synthesis. In *ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 349–356, 2003.

[CF02] Matthew Cooper and Jonathan Foote. Automatic music summarization via similarity analysis. In *International Conference on Music Information Retrieval*, pages 81–85, 2002.

[CJ11] Jeffrey N. Chadwick and Doug L. James. Animating fire with sound. *SIGGRAPH*, 30(4), 2011.

[CKDH00] Ali Taylan Cemgil, Bert Kappen, Peter Desain, and Henkjan Honing. On tempo tracking: Tempogram representation and kalman filtering, 2000.

[Dij55] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Mathematik*, pages 269–271, 1955.

[Dix07] Simon Dixon. Evaluation of the audio beat tracking system beatroot. *Journal of New Music Research*, 36(1):39–50, 2007.

[DPN08] Thomas Deselaers, Lexi Pimenidis, and Hermann Ney. Bag-of-visual-words models for adult image classification and filtering. In *International Conference on Pattern Recognition*, pages 1–4, 2008.

[Epp98] David Eppstein. Finding the k shortest paths. *SIAM J. Computing*, 28(2):652–673, 1998.

[Foo00] Jonathan Foote. Automatic audio segmentation using a measure of audio novelty. In *IEEE International Conference on Multimedia and Expo*, volume 1, pages 452–455, 2000.

[FP02] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002.

[FU01] Jonathan Foote and Shingo Uchihashi. The beat spectrum: A new approach to rhythm analysis. In *ICME*, 2001.

[GL08] S. Grofit and Y. Lavner. Time-scale modification of audio signals using enhanced wsola with management of transients. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(1):106–115, 2008.

[GM09] Peter Grosche and Meinard Müller. A mid-level representation for capturing dominant tempo and pulse information in music recordings. In *ISMIR*, pages 189–194, 2009.

[Góm06] E. Gómez. *Tonal Description of Music Audio Signals*. PhD thesis, Universitat Pompeu Fabra, 2006.

[HNR68] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics*, 4(2):100–107, 1968.

[Jen07] Kristoffer Jensen. Multiple scale music segmentation using rhythm, timbre, and harmony. *EURASIP J. Appl. Signal Process.*, 2007(1):159–159, 2007.

[JR07] J. C. Segura J. Ramirez, J. M. Górriz. Voice activity detection. fundamentals and speech recognition system robustness. *Robust Speech Recognition and Understanding*, pages 1–22, 2007.

[KGP02] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. In *SIGGRAPH*, pages 1–10. ACM, 2002.

[KK97] R. W. L. Kortekaas and A. Kohlrausch. Psychoacoustical evaluation of the pitch-synchronous overlap-and-add speech-waveform manipulation technique using single-formant stimuli. *Acoustical Society of America Journal*, 101:2202–2213, 1997.

[Lar02] Jean Laroche. Time and pitch scale modification of audio signals. In Mark Kahrs and Karlheinz Brandenburg, editors, *Applications of Digital Signal Processing to Audio and Acoustics*, volume 437 of *The Kluwer International Series in Engineering and Computer Science*, pages 279–309. Springer US, 2002.

[LC00] B. Logan and S. Chu. Music summarization using key phrases. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2 of *ICASSP '00*, pages 749–752. IEEE Computer Society, 2000.

[LCOZ+11] Jinjie Lin, Daniel Cohen-Or, Hao Zhang, Cheng Liang, Andrei Sharf, Oliver Deussen, and Baoquan Chen. Structure-preserving retargeting of irregular 3d architecture. *ACM Transactions on Graphics (Proceedings SIGGRAPH Asia)*, 30(6):183:1–183:10, 2011.

[LD99] J. Laroche and M. Dolson. Improved phase vocoder time-scale modification of audio. *IEEE Transactions on Speech and Audio Processing*, 7(3):323–332, 1999.

[Les08] Lawrence Lessig. *Remix : making art and commerce thrive in the hybrid economy*. Bloomsbury, London, 2008.

[LHL10] Sylvain Lefebvre, Samuel Hornus, and Anass Lasram. By-example synthesis of architectural textures. *ACM Transactions on Graphics (SIGGRAPH Conference Proceedings)*, 2010.

[Log00] Beth Logan. Mel frequency cepstral coefficients for music modeling. In *International Symposium on Music Information Retrieval*, 2000.

[Low04] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

[LWB+11] Zhang Liu, Chaokun Wang, Yiyuan Bai, Hao Wang, and Jianmin Wang. Musiz: a generic framework for music resizing with stretching and cropping. In *ACM International Conference on Multimedia*, pages 523–532. ACM, 2011.

[LWZ04] Lie Lu, Liu Wenyin, and Hong-Jiang Zhang. Audio textures: theory and applications. *Speech and Audio Processing, IEEE Transactions on*, 12(2):156–167, 2004.

[LZ03] Lie Lu and Hong-Jiang Zhang. Automated extraction of music snippets. In *ACM International Conference on Multimedia*, pages 140–147, 2003.

[Mac67] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[Mar06] Matija Marolt. A mid-level melody-based representation for calculating audio similarity. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, pages 280–285, 2006.

[Mer76] P. Mermelstein. Distance measures for speech recognition: Psychological and instrumental. In C. H. Chen, editor, *Pattern Recognition and Artificial Intelligence*, pages 374–388. Academic Press, New York, 1976.

[MW10] J. L. Myers and A. D. Well. *Research Design and Statistical Analysis*. Lawrence Erlbaum Associates, New Jersey, third edition, 2010.

[NSGS11] Enrique Bermejo Nievas, Oscar Deniz Suarez, Gloria Bueno García, and Rahul Sukthankar. Violence detection in video using computer vision techniques. In *International Conference on Computer Analysis of Images and Patterns*, pages 332–339, 2011.

[O'S87] Douglas O'Shaughnessy. *Speech communication: human and machine*. Addison-Wesley, 1987.

[PB04] J. R. Parker and B. Behm. Creating audio textures by example: tiling and stitching. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages 317–320, 2004.

[PLR07] Ewald Peiszer, Thomas Lidy, and Andreas Rauber. Automatic audio segmentation: Segment boundary and structure detection in popular music, 2007.

[PMK10] Jouni Paulus, Meinard Müller, and Anssi Klapuri. Audio-based music structure analysis. In *International Conference on Music Information Retrieval*, pages 625–636, 2010.

[RGSS10] Michael Rubinstein, Diego Gutierrez, Olga Sorkine, and Ariel Shamir. A comparative study of image retargeting. *SIGGRAPH Asia*, 29(5):160:1–160:10, 2010.

[Sch11] Diemo Schwarz. State of the art in sound texture synthesis. In *International Conference on Digital Audio Effects*, 2011.

[SERlG06] G. Strobl, G. Eckel, D. Rocchesso, and S. le Grazie. Sound texture modeling: A survey. pages 61–5, 2006.

[SM00] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, 2000.

[SNI06] Takaaki Shiratori, Atsushi Nakazawa, and Katsushi Ikeuchi. Dancing-to-music character animation. *Computer Graphics Forum*, 25(3):449–458, 2006.

[SSSE00] Arno Schödl, Richard Szeliski, David H. Salesin, and Irfan Essa. Video textures. In *SIGGRAPH*, pages 489–498, 2000.

[TBPM05] Manolis Terrovitis, Spiridon Bakiras, Dimitris Papadias, and Kyriakos Mouratidis. Constrained shortest path computation. *Symposium on Large Spatial Databases*, pages 181–199, 2005.

[Ver00] Werner Verhelst. Overlap-add methods for time-scaling of speech. *Speech Commun.*, 30(4):207–221, 2000.

[Vit67] Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13(2):260–269, 1967.

[VR93] Werner Verhelst and Marc 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*, pages 554–557, 1993.

[WM11] Stephan Wenger and Marcus Magnor. Constrained example-based audio synthesis. In *IEEE International Conference on Multimedia and Expo*, 2011.

[XZT02] Changsheng Xu, Yongwei Zhu, and Qi Tian. Automatic music summarization based on temporal, spectral and cepstral features. In *IEEE International Conference on Multimedia and Expo*, volume 1, pages 117–120, 2002.

[Zie07] Mark Ziegelmann. *Constrained Shortest Paths and Related Problems - Constrained Network Optimization*. VDM Verlag, 2007.

[ZJ10] Changxi Zheng and Doug L. James. Rigid-body fracture sound with precomputed soundbanks. *SIGGRAPH*, 29(3), 2010.