Pitch Scaling of Music Signals

Théo Verhelst
Université Libre de Bruxelles

Abstract

Pitch scaling is the task of modifying the frequency of a signal while keeping its playback speed intact. Pitch scaling is an important feature of many digital music tools, and needs to be done in real-time with the highest available quality. In this report, we describe succinctly state-of-the-art techniques of pitch scaling, and we explain in detail one that is particularly suited to music signals. We also describe a Clojure implementation of this technique. This technique uses phase vocoder for time-scaling and 3rd order spline interpolation for resampling.

Introduction

Pitch scaling is a process that changes the frequency of a signal without modifying its speed. The main difficulty of pitch scaling is to make the synthesised signal as natural as the original, so that it sounds like as if it was recorded or created on this new pitch. More precisely, the timbre and the speed of the signal need to be preserved.

One application of pitch scaling is to scale the pitch of an instrument recording in a music production software, in order to tune it to the other instruments, or for any other musical purpose. But the main use of pitch scaling is probably in DJ software (Cliff and Cliff, 2000): one can change the pitch of a track, and therefore its key, in order to make the transition to the next track easier and smoother.

The most straightforward way to change the pitch of an audio signal is to resample the signal and playing it back at its original rate. But both pitch and speed are modified at the same time. Thus, we need a more sophisticated technique in order to preserve the playback speed constant (Driedger and Müller, 2015).

In contrast to pitch scaling, time-scale modification (TSM) is a process that modifies the speed of a signal without modifying its pitch and its timbre. TSM has been subject to many more studies than pitch scaling, but we base our work on these studies, since it is possible to show that both processes are mathematically equivalent. Indeed, in order to change the pitch of a signal, one can use a well-known TSM method, and then resample the signal.

Time-scale modification techniques

TSM techniques can be grouped in two main categories: *time-domain TSM* and *frequency-domain TSM*.

Time-domain TSM extracts portions of the input signal at a frequency defined by the scaling factor, and places them in the output signal. This technique is particularly suited to monophonic, harmonic signal, as it preserves almost perfectly the timbre of the signal. The basic time-domain algorithm, *overlap-and-add* (or *OLA*), suffers from phase jumps artifacts, but there are many variations of this algorithm that avoid this effect. But all of them are only able to correct phase jumps on the most prominent frequency, i.e. the fundamental frequency of the signal. Phase jumps can still occur in less important frequencies, i.e. the harmonics, which is clearly audible. OLA-base algorithms are thus not suited to polyphonic signals, since they contains more harmonics than monophonic ones. Furthermore, non-harmonic signals, such as drums or percussive instruments, have non-periodic patterns. These patterns are known as *transients*. A typical time-domain TSM technique leads to transient doubling or skipping (depending on the scaling factor), since these techniques periodically repeat or discard some small parts of the signal. This can be reduced by taking a very short frame size, or managing separately the transients after a transient detection procedure (Grofit and Lavner, 2008).

Frequency-domain TSM is based on the short-time Fourier transform (STFT). It splits the signal in small chunks, and computes the Fourier transform of each of these chunks, in order to get a discrete frequency-domain representation of the signal. Often, the technique also uses the *phase vocoder* in order to refine the frequencies estimates, and are thus named *phase-vocoder time-scale modification*, or *PV-TSM*, or simply phase vocoder. The idea is to preserve phase continuity across all frequencies, and not only on the most prominent frequency as in time-domain TSM, by exploiting the frequency-domain representation of the sound. PV-TSM behaves well on polyphonic signals, but are subject to vertical phase incoherence, i.e. the relationship between the phases of different frequencies at a point of time is not preserved, leading to audible artifacts, known as *phasiness*, or *loss of presence*.

General procedure

Pitch scaling process is split in two steps: 1) apply a TSM procedure 2) resample the signal. Let α be the scaling factor. We first apply a TSM procedure with parameter α, so that the playback speed is modified, while the pitch is left unmodified. Then, we resample the signal by a factor $1/\alpha$, so that the playback speed of the signal is the same as the original, but the pitch is multiplied by α.

For the TSM part, we use a simple phase vocoder with phase propagation. For the resampling part, we use an interpolator based on 3rd order splines. These are explained below.

Time-scale modification

Basics of time-scale modification

Notation. Let

$$[a:b] := \{a, a+1, \dots, b-1, b\} \quad \forall a, b \in \mathbb{Z} : a < b$$

and

$$[a:b[:= \{a, a+1, \dots, b-1\} \quad \forall a, b \in \mathbb{Z} : a < b$$

First, we have to define the basic concepts involved in time-scaling. Let the function $x : \mathbb{Z} \to [-1, 1]$ be signal to time-scale. In practice, the analysed audio signal has a finite duration of $L \in \mathbb{N}$ samples. Thus we define $x(n) = 0 \, \forall n \in \mathbb{Z} \setminus [0 : L[$ for the sake of simplicity. We want to construct the output signal $y : \mathbb{Z} \to [-1, 1]$ that have the same frequency-domain properties as x, but being stretched in time by the factor α.

Almost all TSM techniques are based on the following procedure: first, x is divided in *analysis frames* x_m, $m \in \mathbb{Z}$ having each a length of N samples, and these analysis frames are spaced by an *analysis hop size* H_a:

$$x_m(n) = \begin{cases} x(mH_a + n) & \text{if } n \in [0 : N[\\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Then, we could want to put these frames in the output signal by spacing them by the synthesis hop size H_s, achieving the correct time-scale modification. But this would cause very audible phase discontinuities, since the end of a frame would no longer match with the beginning of the next frame. Thus, we need to modify the analysis frames x_m into *synthesis frames* y_m before adding them to the output signal $y(n)$:

$$y(n) = \sum_{m \in \mathbb{Z}} y_m(n - mH_s) \tag{2}$$

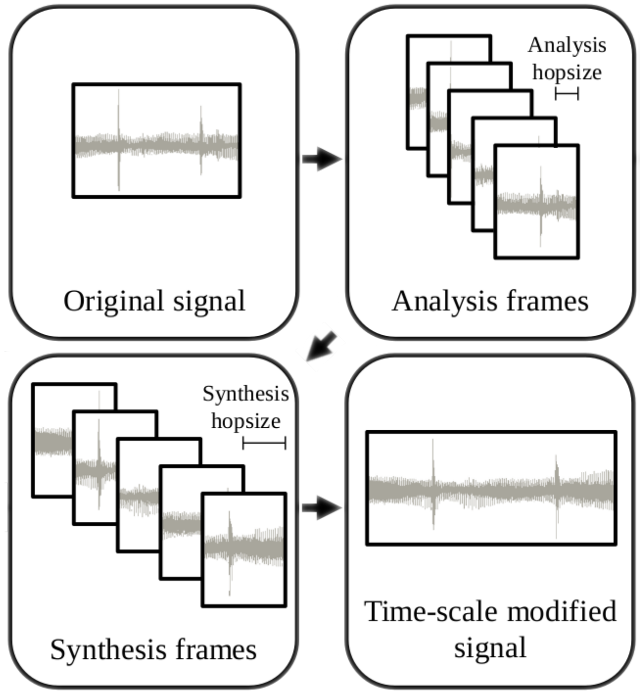

Figure 1: General procedure of time-scale modification.

H_s is usually set to $N/2$ or $N/4$, in order to have a constant overlap between the synthesis frames. And since we know that $\alpha = \frac{H_s}{H_a}$, we then have $H_a = \frac{H_s}{\alpha}$.

The method used to transform x_m into y_m is critical, as it determines the quality of the result. In addition to phase discontinuities, it must also compensate gain fluctuations: if no care is taken, two overlapping frames may have a higher gain in the overlapping part than in the rest of the signal. A common procedure to compensate gain fluctuations is to multiply each modified frame by a windowing function $w : \mathbb{N} \to [0, 1]$ before adding them to the output signal y:

$$y(n) = \sum_{m \in \mathbb{Z}} w(n - mH_s) y_m(n - mH_s) \tag{3}$$

The Hann window is widely used for this purpose, and is defined as

$$w(n) = \begin{cases} 0.5(1 - cos(\frac{2\pi n}{N-1})) \text{ if } n \in [0 : N[\\ 0 \text{ otherwise} \end{cases} \tag{4}$$

The Hann window has the property that if we add Hann windows spaced by $N/2$ in their domain, the sum of the overlapping parts we be equal to one:

$$\forall n \in \mathbb{Z} \quad \sum_{i \in \mathbb{Z}} w(n + i\frac{N}{2}) = 1 \tag{5}$$

This property is essential to the equation (3): if not satisfied, the output signal may be multiplied by more than one in some places.

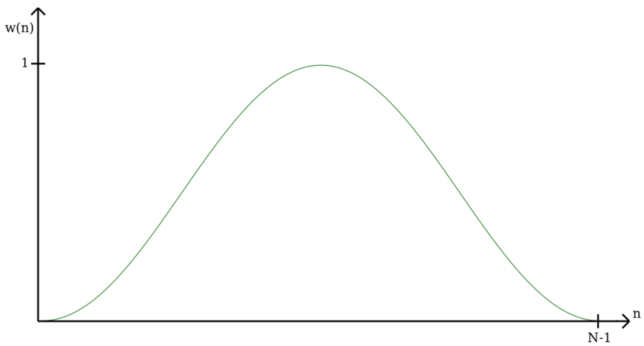

Figure 2: The Hann window function.

Phase-vocoder time-scale modification

The phase vocoder is a common frequency-domain technique used to derive the *synthesis frames* y_m from the analysis frames x_m. The idea is to compute the spectrum of the frame with the Fourier transform, and change the phases of this spectrum so that the signal re-synthesised from this modified spectrum has no phase jump with the following and previous frames.

Short-time Fourier transform The spectrum of a signal is the frequency-domain representation of a time-domain signal, and is composed by the amplitudes and phases of the Fourier series decomposition of the signal. The Fourier transform of discrete-time signals is primarily defined for signals of length N (Bracewell and Bracewell, 1986):

$$X(k) = \sum_{n=0}^{N-1} x(n) \exp(-2\pi ikn/N) \tag{6}$$

where $k \in [0 : N[$ denotes the frequency index (see (10)). This yields k complex numbers, where $|X(k)|$ is the magnitude of the kth frequency, and $\arg(X(k)) \in [0, 1[$ is the phase of the kth frequency. Note that we use phases in the unit range because we also manipulate frequencies in Hertz, and we don't want to divide the phase by 2π each time we use it in an equation involving a frequency. Thus, we have

$$X(k) = |X(k)| \exp(2\pi \mathrm{i} \arg(X(k))) \tag{7}$$

As required by the time-scaling algorithm, we will apply the Fourier transform on small portions of the signal. We could want to do apply directly the transform on analysis frames x_m:

$$X(m, k) = \sum_{n=0}^{N-1} x_m(n) \exp(-2\pi ikn/N)$$

But this can lead to unexpected high amplitudes in the high frequencies: the Fourier transform works as if the signal was periodic with a period N, i.e. as if we where computing the Fourier series of a signal

$$\tilde{x} : \mathbb{Z} \to \mathbb{R} : n \mapsto x_m(n \mod N) \tag{8}$$

which can have high amplitude in high frequencies around $n = aN \ \forall a \in \mathbb{Z}$. A solution is to apply a windowing function to the analysis frame, so that the signal is always zero at the frame boundaries (Gabor, 1946):

$$X(m, k) = \sum_{n=0}^{N-1} x_m(n) w(n) \exp(-2\pi ikn/N) \tag{9}$$

$X(m, k)$ is the coefficient of the *short-time Fourier transform* of the signal x at time m and at frequency k. It is also named a *time-frequency bin*. w is a windowing function, usually the Hann window as defined in (4)

Note that the frequency index k corresponds to the physical frequency

$$F_{\text{coef}}(k) = \frac{F_s k}{N} \tag{10}$$

and the frame index m corresponds to the physical time

$$T_{\text{coef}}(m) = \frac{H_a m}{F_s} \tag{11}$$

Synthesis with modified frames Our goal is to compute the synthesis time-frequency bins $Y(m, k)$ from the analysis time-frequency bins $X(m, k)$, so that the signal synthesised with the inverse Fourier transform from these synthesis time-frequency bins has phase coherence from one frame to the next. Since we are only interested in phase adjustment, we can already set

$$|Y(m, k)| = |X(m, k)| \tag{12}$$

We also have to determine the phase $\arg(Y(m, k))$, so that we can synthesise the synthesis frame y_m from the synthesis time-frequency bins $Y(m, k)$. Note that the modified STFT may not be invertible, i.e. there may be no signal y whose STFT is $Y(m, k)$. However, there is a procedure described in (Griffin et al., 1984) that minimizes the squared error between $Y(m, k)$ and the STFT of the resulting signal. It results that the synthesis frame y_m is given by:

$$y_m(n) = \frac{1}{N} \sum_{k=0}^{N-1} Y(m, k) \exp(2\pi \mathrm{i} kn/N) \tag{13}$$

We can then reconstruct the output signal y with the equation (3).

Insert graphic on inst frequency here

Instantaneous frequency The phase adjustment involves the observation that the short-time Fourier transform yields coefficients for a finite quantity of frequencies, and that may be insufficient to characterise exactly the underlying sinusoids involved in the Fourier series decomposition of the signal. However, it is possible to refine the frequency of the kth frequency index by using the phases of the current and next frame. This refined frequency is called the *instantaneous frequency* and is denoted by $F_{\text{coeff}}^{\text{IF}}(m, k)$.

Notation. Let

$$\phi_m := \arg(X(m, k))$$

and

$$\gamma_m := \arg(Y(m, k))$$

In order to find the instantaneous frequency, we first can observe that for any time-frequency bin $X(m, k)$, we can compute its *unwrapped phase advance*, that is, the phase increment that should occur from the time $T_{\text{coef}}(m)$ to the time $T_{\text{coef}}(m + 1)$, according to the frequency estimate $F_{\text{coef}}(k)$:

$$\phi^{\text{inc}} = F_{\text{coef}}(k)\Delta t_a \tag{14}$$

where Δt_a is analysis hop time given is seconds:

$$\Delta t_a = H_a/F_s \tag{15}$$

Since we know the phase ϕ_m, we can compute the predicted phase at the time $T_{\text{coef}}(m)$:

$$\phi_m^{\text{pred}} = \phi_m + \phi^{\text{inc}} \tag{16}$$

But because of the lack of precision of the phase vocoder, this predicted phase may not be equal to ϕ_{m+1} when mapped in the range $[0, 1[$. We can then compute the difference with the effective phase ϕ_{m+1}, and this difference is called the *phase error* (or *heterodyned phase increment*):

$$\phi_m^{\text{err}} = \Psi(\phi_{m+1} - \phi_m^{\text{pred}}) \tag{17}$$

where the function Ψ is the *principal argument function* that maps the phase difference to the range $[-0.5, 0.5]$. Here is a possible implementation of Ψ:

$$\Psi : \mathbb{R} \to [-0.5, 0.5] : \phi \mapsto \phi - \lceil \phi - 0.5 \rceil \tag{18}$$

From this phase error, we can compute the *instantaneous frequency*: this is a refinement of $F_{\text{coef}}(k)$, in an attempt to determine the frequency of the underlying sinusoid by taking the phase error into account. This sinusoid should have a phase of ϕ_m at the time $T_{\text{coef}}(m)$ and a phase ϕ_{m+1} at the time $T_{\text{coef}}(m + 1)$.

$$\begin{aligned} F_{\text{coeff}}^{\text{IF}}(m, k) &= \frac{\phi^{\text{inc}} + \phi_m^{\text{err}}}{\Delta t_a} && (19) \\ &= F_{\text{coef}}(k) + \frac{\phi_m^{\text{err}}}{\Delta t_a} && (20) \\ &= F_{\text{coef}}(k) + \frac{\Psi(\phi_{m+1} - \phi_m - \phi^{\text{inc}})}{\Delta t_a} && (21) \end{aligned}$$

Phase adjustment Now that we have refined the coarse frequency estimates of the STFT, we can use this information to adjust the phase of the time-frequency bins. The idea is to set the phase of the bin according to the phase and the instantaneous frequency of corresponding bin in the previous frame:

$$\gamma_m = \gamma_{m-1} + F_{\text{coeff}}^{\text{IF}}(m - 1, k)\Delta t_s \tag{22}$$

We now use the synthesis hop time since we are synthesising the output frames, and they are spaced by the synthesis hoptime $\Delta t_s = H_s/F_s$.

This formula recursively defines the modified frame from the previous modified frame and a refined frequency estimate of the underlying sinusoid. Thus, we need an initial step for the first frame. In practice, we do not change the phases of the first frame, since there is no phase discontinuity with the previous frame:

$$\gamma_0 = \phi_0 \tag{23}$$

Now that we have a synthesis time-frequency bin for the whole signal, we can use the equation (13) to construct the synthesis frames, and then equation (3) to synthesise the final output signal.

Result and limitations

Thanks to the phase modification procedure, the phase continuity is guaranteed in successive frames, and allows efficient time-scaling of a polyphonic musical signal. This phase phase continuity in time is called horizontal phase coherence. But the phase relationship between the frequency channels in one frame is no longer respected. This relationship is called vertical phase coherence. This leads to phasiness, and the resulting sound is characteristic of phase-vocoder techniques (Laroche and Dolson, 1999). Another effect is transient smearing: in musical signals with percussions, such as drums, the percussive aspect of the sound results of the subtle and sudden tuning of the phases of various sinusoids in the signal. And because of the lack of vertical phase coherence, the percussive aspect of transients is significantly reduced.

Further improvements

Many improvements has been proposed to the standard phase-vocoder procedure, mainly to reduce the phasiness and transient smearing of the result. One of them (Dorran et al., 2006; Kraft et al., 2012), is based on the observation that the phasiness increases over time from the time where the modified phase is set as the initial input phase (see (23)). Thus, by placing periodically in the output signal an analysis frame unchanged, the vertical phase coherence is reset before it degenerates too much. The unmodified frame is placed in the signal with usual time-domain procedure, such as *Waveform Similarity Overlapp and Add* (*WSOLA*) (Verhelst and Roelands, 1993).

Another well-known improvement of the standard phase-vocoder has been proposed by J. Laroche and M. Dolson (Laroche and Dolson, 1999). It is based on the observation that the phase of a sinusoid is mainly tied to the phase of the nearest peak in the spectrum. Although phase inconstency between a very low-frequency sinusoid and a high-pitched one is barely noticeable, a high-amplitude sinusoid influences the phase of nearby frequency channels. Garbaging this latter relation results in audible phasiness. The idea of this improved phase-vocoder is to lock the phase of all frequency channels to the one of the neared peak in the spectrum. As a result, the vertical phase incoherence is greatly reduced. Furthemore, the computational cost of the phase-vocoder is also reduced since the phase propagation has to be computed only for the peak frequencies.

Resampling

The goal of resampling is to reconstruct a continuous-time signal from the given discrete-time samples, and then sample this signal again with another sampling rate. More formally, we want to construct the continuous-time signal

$$\hat{x} : \mathbb{R} \to \mathbb{R} \tag{24}$$

such that

$$x(n) = \hat{x}(Tn)\ \forall n \in \mathbb{Z} \tag{25}$$

where T is the sampling period (the inverse of the sampling rate). Then, we sample a new signal y at a sampling period T':

$$y(n) = \hat{x}(T'n)\ \forall n \in \mathbb{Z}$$

Procedure

In order to construct the continuous-time signal, we need an interpolator. There exists various interpolators, such as truncated sinc (Duncan and Rossum, 1988; Rossum, 1989), linear-interpolator (Rossum, 1993), b-spline interpolator (Sankar and Ferrari, 1988), Lagrange interpolator (Schafer and Rabiner, 1973). We chose 3rd order Hermitian spline interpolator, for its simple implementation (Grisoni et al., 1997).

For 3rd order spline interpolation, we search the factors of a 3rd-degree polynomial for each pair of consecutive signal values $(x(n),\, x(n+1))$, such that this polynomial passes through these values.

Notation. Let

$$y_0 = x(n-1);\ y_1 = x(n);\ y_2 = x(n+1);\ y_3 = x(n+2)$$

We search a function

$$f : [0,1] \to \mathbb{R} : t \mapsto \sum_{i=0}^{3} \alpha_i t^i = \alpha_0 + \alpha_1 t + \alpha_2 t^2 + \alpha_3 t^3 \tag{26}$$

In order to determine the value of the factors α_i, we need to set four constraints on the function f:

$$\begin{cases} f(0) = y_1 & (27) \\ f(1) = y_2 & (28) \\ f'(0) = \dfrac{y_2 - y_0}{2} & (29) \\ f'(1) = \dfrac{y_3 - y_1}{2} & (30) \end{cases}$$

(27) and (28) are natural constraints of spline interpolator: we want that the interpolating function passes through the supplied values $(x(n),\, x(n+1))$. But in order to determine the factors α_i, we need two more constraints. One solution is to use Hermitian splines (De Boor et al., 1978), that is, the derivative of f at $t \in \{0,1\}$ is equal to the derivative of a straight line between the previous and the next point ((29) and (30)). By expressing the equations as a matrix equation, we can find that

$$\begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ -3 & 3 & -1 & -0.5 \\ 2 & -2 & 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_2 - y_0 \\ y_3 - y_1 \end{bmatrix} \tag{31}$$

For each pair of samples, we just have to compute the four factors α_i with this equation, and then evaluate the function f as in (26), at a value of t corresponding to the new sampling rate.

Results and limitations

By following the described resampling procedure, one can achieve fairly good results. Only informal tests have been done, but no audible artifact has been noticed, and the result is close to what one would expect from any resampling algorithm.

Although the input signal has the domain $[-1, 1]$, this resampling technique does not guarantee that the resampled signal will also fit in this range. For example, if we have $y = \begin{bmatrix} 1 & -1 & -1 & 1 \end{bmatrix}^T$, it follows that we have the factor vector $\begin{bmatrix} -1 & -1 & 1 & 0 \end{bmatrix}^T$, which results in the function

$f : t \mapsto -1 - t + t^2$. If we evaluate this function at $t = 0.5$, we have $f(0.5) = -1.25$. This may result in gain fluctuation, or even in distortion or clipping of the signal, which is clearly not ideal.

Implementation

Functionnal programming

Although the main objective of this thesis is to accomplish a research work on a scientific subject, another one is to discover the functionnal paradigm for programming languages, and the functionnal language *Clojure*. Functionnal programming constrasts with the widely-used imperative paradigm. It is aimed to build stronger softwares, based on the concept of *pure function*. A functions is said to be pure if it is free of any noticeable side-effect, and if it always returns the same result for a given set of arguments. Common mathematical functions, such as sin, are typical examples of pure function, whereas the usual `print` function is not, since it outputs characters to some output device.

Functionnal purity improves reliability of software by ensuring that when a pure function is called, all the consequences of this call are known by just looking at the caller code. Obviously, not all function are suited to be pure. But the more the code is functionnaly pure, the more the software is side-effect free, and consequently less bug-prone.

The Clojure programming language

Clojure is based on the *Lisp* programming language, which dates back to 1958. Like Lisp, it offers fully-parenthesised syntax, and is based on the *Lambda calculus* (Rojas, 1998).

In order to manipulate audio data in Clojure, the music library *Overtone* is used. It allows a very large range of features for music manipulation and generation, through the use of *generators*. A generator is a function object that can be used to generate sound, or to control the value of a parameter over the time. A generator can also be parametrized by other generator, allowing a great flexibility and creativity in its design.

The goal of the implementation in Clojure is to show the various concepts elaborated earlier, in the light of the functional programming paradigm. As a result, we used a very small portion of the Overtone library, mainly the WAV file read and write procedures. We could have implemented the phase vocoder in a few lines only with Overtone, but the juicy part of the algorithm would have been hidden behind a library function call. We then choosen to mainly use the built-in features of the Clojure language, in order to make the whole procedure explicit.

Implementation of pitch scaling

The pitch-shifting implementation is split in three main parts, each having its own namespace: the time-scaling algorithm in the namespace `time-scaling` , the resampling algorithm in the namespace `resampling`, and the pitch-shifting algorithm in the namespace `pitch-shifting`. The latter uses the two former, as discussed in the general procedure section. The front-end of the library is the namespace `core`, which contains the `-main` function. This function parses the command-line arguments and apply the pitch-shifting algorithm.

Each algorithm is clearly separated from the others, achieving a good code independance. This enables an interesting feature for this library: because the resampling is separated from the time-scaling, we could use two different parameters for both algorithm. This allows us to change the pitch and the duration of the audio signal at the same time, without any additional computational cost. Given that one of the goal of this library is to provide a convenient tool for DJing set preparation, this allows the user to change the tempo and the key of a music file in only one operation, in order to make a smooth transition with another music file.

Results

Conclusion

We presented state-of-the-art pitch-shifting procedure suited to music signals, based on the standard phase-vocoder technique and on resampling with 3rd-order spline interpolation. This lead to fairly good results, although there is still room for improvement, both on quality and on efficiency. We then briefly introduced the functionnal programming paradigm, as well as the Clojure programming language. Finally, we described our implementation of the pitch-shifting algorithm in Clojure.

References

Bracewell, R. N. and Bracewell, R. N. (1986). *The Fourier transform and its applications*, volume 31999. McGraw-Hill New York.

Cliff, D. and Cliff, D. (2000). Hang the dj: Automatic sequencing and seamless mixing of dance-music tracks. Technical report, HP Labs Technical Report.

De Boor, C., De Boor, C., Mathématicien, E.-U., De Boor, C., and De Boor, C. (1978). *A practical guide to splines*, volume 27. Springer-Verlag New York.

Dorran, D., Coyle, E., and Lawlor, R. (2006). Audio time-scale modification using a hybrid time-frequency domain approach.

Driedger, J. and Müller, M. (2015). A review of time-scale modification of music signals. In Valimaki, V., editor, *Proceedings of the International Conference on Digital Audio Effects (DAFx)*.

Duncan, A. and Rossum, D. (1988). Fundamentals of pitch-shifting. In *Audio Engineering Society Convention 85*.

Gabor, D. (1946). Theory of communication. part 1: The analysis of information. *Electrical Engineers - Part III: Radio and Communication Engineering, Journal of the Institution of*, 93(26):429–441.

Griffin, D. W., Jae, Lim, S., and Member, S. (1984). Signal estimation from modified short-time fourier transform. *IEEE Trans. Acoustics, Speech and Sig. Proc*, pages 236–243.

Grisoni, L., Schlick, C., and Blanc, C. (1997). An hermitian approach for multiresolution splines. Technical report.

Grofit, S. and Lavner, Y. (2008). 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.

Kraft, S., Holters, M., von dem Knesebeck, A., and Zölzer, U. (2012). Improved pvsola time-stretching and pitch-shifting for polyphonic audio.

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

Rojas, R. (1998). A tutorial introduction to the lambda calculus.

Rossum, D. (1989). An analysis of pitch-shifting algorithms. In *Audio Engineering Society Convention 87*.

Rossum, D. (1993). Constraint based audio interpolators. In *Proceedings of IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 161–164.

Sankar, P. V. and Ferrari, L. A. (1988). Simple algorithms and architectures for b-spline interpolation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10(2):271–276.

Schafer, R. W. and Rabiner, L. R. (1973). A digital signal processing approach to interpolation. *Proceedings of the IEEE*, 61(6):692–702.

Verhelst, W. and Roelands, M. (1993). An overlap-add technique based on waveform similarity (wsola) for high quality time-scale modification of speech. In *1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 554–557 vol.2.