

Chapter 5

How does an audio effects processor perform pitch-shifting?

"Christmas, Christmas time is near, Time for toys and time for cheer, ..."
The chipmunk song (Bagdasarian 1958)

T. Dutoit (°), J. Laroche (*)

(°) Faculté Polytechnique de Mons, Belgium

(*) Creative Labs, Inc., Scotts Valley, California

The old-fashioned charm of modified high-pitched voices has recently been revived by the American movie *Alvin and the Chipmunks*, based on the popular group and animated series of the same name which dates back from the fifties. These voices were originally performed by recording speech spoken (or sung) slower (typically at half the normal speed), and then accelerating ("pitching-up") the playback (typically at double speed, thereby increasing the pitch by one octave). The same effect can now be created digitally and in real time; it is widely used today in computer music.

5.1 Background – The phase vocoder

It has been shown in Chapter 3 (3.1.2 and 3.2.4) that linear transforms and their inverse can be seen as sub-band analysis/synthesis filter banks. The main idea behind the *phase vocoder*, which originally appeared in (Flana-

gan and Golden 1966¹⁾ for speech transmission, is precisely that of using a discrete Fourier transform (DFT) as a filter bank for sub-band processing of signals (5.1.1). This, as we shall see, is a particular case of the more general Short-Time Fourier Transform (STFT) processing (5.1.2) with special care to perfect reconstruction of the input signal (5.1.3), which opens the way to non-linear processing effects, such as time-scale modification (5.1.4), pitch-shifting (5.1.5), harmonization, etc.

5.1.1 DFT-based signal processing

The most straightforward embodiment for frequency-based processing of an audio signal $x(n)$ is given in Fig. 5.1 and Fig. 5.2. Input frames \mathbf{x} containing N samples $[x(mN), x(mN+1), \dots, x(mN+N-1)]$ are used as input for an N -bin DFT. The N resulting complex frequency-domain values $\mathbf{X}=[X_0(m), X_1(m), \dots, X_{N-1}(m)]$ are given by:

$$X_k(m) = \sum_{n=0}^{N-1} x(n+mN)e^{-jn\Omega_k} \quad \text{with } \Omega_k = k \frac{2\pi}{N} \quad (5.1)$$

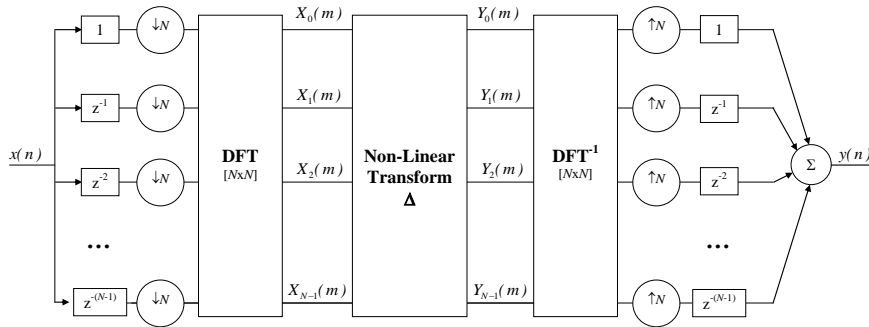


Fig. 5.1 N -bin DFT-based signal processing: block diagram

¹ Flanagan and Golden called this system the *phase vocoder* after H. Dudley's *channel vocoder*, which only transmitted the spectral envelope (i.e. a gross estimation of the amplitude spectrum) of speech, while spectral details were sent in a separate channel and modeled through a voiced/unvoiced generator. In contrast, in the phase vocoder, phase information was sent together with amplitude information, and brought a lot more information on the speech excitation.

They are changed into $\mathbf{Y}=[Y_0(m), Y_1(m), \dots, Y_{N-1}(m)]$ according to the target audio effect Δ , and an inverse DFT produces output audio samples $\mathbf{y}=[y(mM), y(mM+1), \dots, y(mM+N-1)]$.

As already explained in Chapter 3, $X_k(m)$ can also be interpreted, for a fixed value of k , as the result of passing the input signal through a sub-band filter $H_k(z)$ whose frequency response $H_k(\varphi)$ is centered on Ω_k and whose impulse response is given by:

$$h_k(n) = \{1, e^{-j\Omega_k}, e^{-j2\Omega_k}, \dots, e^{-j(N-1)\Omega_k}, 0, 0, \dots\} \quad (5.2)$$

and further decimating its output by a factor N (Fig. 5.3). These sub-band processing and block-processing views are very complimentary (Fig. 5.4) and will be used often in this Chapter.

Notice that in this first simple embodiment, the input and output frames do not overlap: all sub-band signals $X_k(m)$ ($k=0, 1, \dots, N-1$) are implicitly downsampled by a factor N compared to the sampling rate of the input signal $x(n)$.

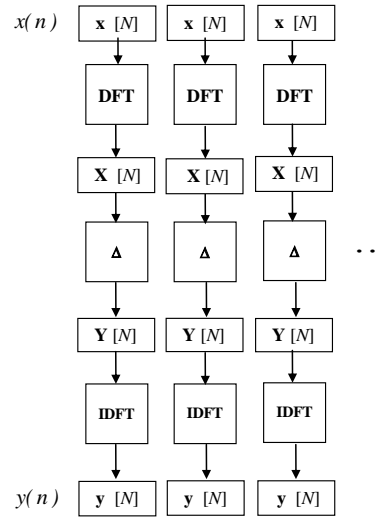


Fig. 5.2 N -bin DFT-based signal processing: frame-based view

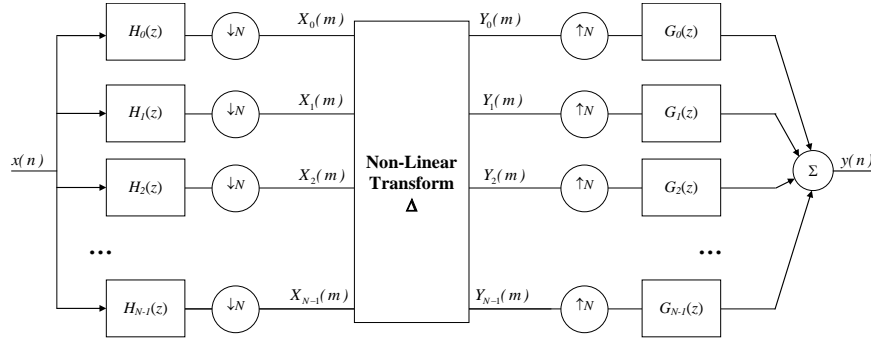


Fig. 5.3 N -channel DFT-based signal processing: sub-band processing view

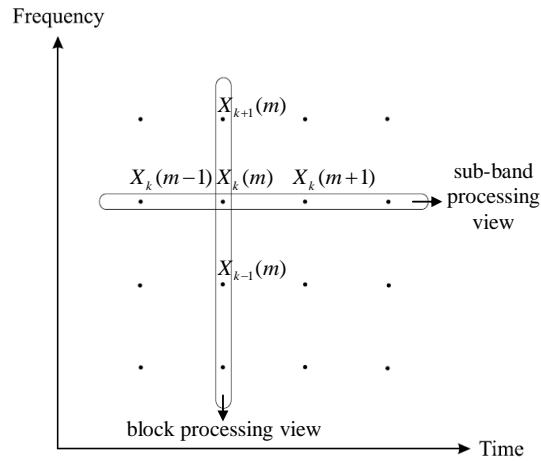


Fig. 5.4 Block processing vs. sub-band signal processing views, in the time-frequency domain (after Dolson, 1986). The former examines the outputs of all sub-band filters at a given instant, while the latter focuses on the output of a single sub-band filter over time.

5.1.2 STFT-based signal processing

The setting depicted in Fig. 5.1 has three apparent problems for the type of non-linear processing we will apply to sub-band signals in this Chapter. These problems are respectively related to frequency selectivity, frame

shift, and to the so-called *blocking effect*. Solving them implies, as we shall see, to use analysis weighting window, overlapping, and synthesis weighting windows.

Analysis weighting windows

DFT filter banks are not frequency selective, as already shown in Fig 3.19, which reveals important side lobes in the frequency response of each filter. This is a serious problem for the design of a phase vocoder: as we will see in Section 5.1.4, the processing applied to each sub-band signal in a phase vocoder is based on the hypothesis that, when the input signal is composed of sinusoidal components (called *partials*, in computer music), each sub-band signal will depend on at most one partial.

A solution is to use an analysis weighting window before the DFT². As a matter of fact, the frequency response of the filter banks implemented by such a weighted DFT is the Fourier Transform of the weighting window itself. Using windows such as Hanning, Hamming, Blackman, or others, makes it possible to decrease the level of side lobes, at the expense of enlarging the bandwidth of each sub-band filter (Fig. 5.5). This drawback can easily be compensated by increasing N , the number of samples in each input frame.

Overlapping analysis frames

In Fig. 5.2, the sampling period of each sub-band signal, i.e. the shift M between successive DFT frames y , was explicitly set to N samples, while the normalized bandwidth of each sub-band signal $X_k(m)$ is approximately equal to $2/N$ (the width of the main lobe of the DFT of the weighting window, which is implicitly rectangular in Fig. 5.2). According to Shannon's theorem, we should have:

$$\begin{aligned} \frac{F_e}{M} &\geq B \\ M &\leq \frac{F_e}{B} \end{aligned} \tag{5.3}$$

where B/F_e is the normalized bandwidth of the weighting window. To avoid aliasing, sub-band signals should thus be sampled with a sampling period smaller than $N/2$ samples, i.e., input DFT frames should have an overlap greater than $N/2$ (Fig. 5.6).

² Notice that weighting windows are not *always* necessary in STFT-based processing. For FFT-based linear time-invariant filtering, for instance, overlapping rectangular windows do the job (Smith 2007).

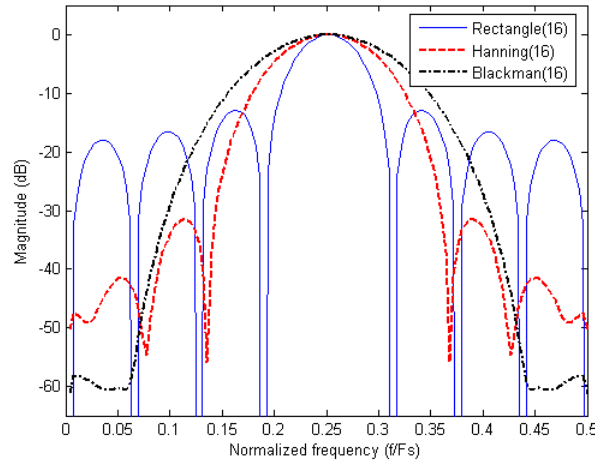


Fig. 5.5 Discrete time Fourier transform of various weighting windows of length $N=16$ samples, multiplied by a sine wave at $1/4^{\text{th}}$ of the sampling frequency. The main lobe width of the rectangular, Hanning, and Blackman windows is respectively equal to $2/N$, $\approx 4/N$, and $\approx 6/N$; side lobes are respectively 13 dB, 31 dB, and 57 dB lower than the main lobe.

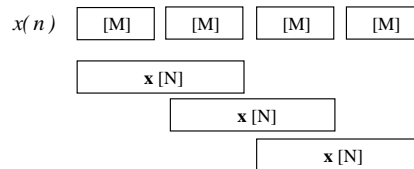


Fig. 5.6 Analysis frames with a frame shift $M=N/2$ (to be compared to the top of Fig. 5.2)

In practice though, we know that if we apply no modification to sub-band signals, i.e. if we set $Y_k(m) = X_k(m)$, the output signal $y(n)$ will be strictly equal to the input signal $x(n)$. This paradox is only apparent: in practice, each sub-band signal is indeed aliased, but the aliasing between adjacent sub-bands cancels itself, as shown in our study of perfect reconstruction filters (3.1.1). This, however, only strictly holds *provided no processing is applied to sub-band signals*.

In contrast, given the complex sub-band signal processing performed in the phase vocoder, it is safer to set the frame shift M in accordance with the Shannon theorem, i.e. to a maximum value of $N/2$ for a rectangular window, $N/4$ for a Hanning window, and $N/6$ for a Blackman window. Besides, we will see in 5.1.4 that the hypotheses underlying sub-band processing in the phase vocoder impose the same constraint.

This transforms our initial DFT (5.1) into the more general *short-time Fourier transform* (STFT):

$$X_k(m) = \sum_{n=0}^{N-1} x(n+mM)w(n)e^{-jn\Omega_k} \quad (k=0, 1, \dots, N-1) \quad (5.4)$$

Synthesis weighting windows

Since the analysis is based on overlapping input frames, synthesis is performed by summing overlapping output frames (as in Fig 3.7). However, when non-linear *block-adaptive* sub-band processing is performed, as in the phase vocoder, processing discontinuities appear at synthesis frame boundaries. This has already been mentioned in 3.1.2.

A partial³ solution to this problem is to use *synthesis weighting windows* before adding overlapping output frames. The resulting processing scheme is termed as *Weighted OverLap-Add* (WOLA):

$$y(n) = \sum_{m=-\infty}^{\infty} y_m(n-mM)w(n-mM) \quad (5.5)$$

$$\text{with } y_m(n) = \frac{1}{N} \sum_{k=0}^{N-1} Y_k(m)e^{jn\Omega_k}$$

where in practice the number of terms in the first summation is limited by the finite length N of the weighting window $w(n)$. The association of IFFT and WOLA is sometimes referred to as *Inverse short-time Fourier transform* (ISTFT).

Fig. 5.7 shows the final block-processing view for STFT-based signal processing: an STFT-based analysis block, whose output (amplitudes and phases) is processed and sent to an ISTFT block. The corresponding sub-band processing view is not repeated here: it is still that of Fig. 5.3 pro-

³ The use of synthesis weighting windows only spreads discontinuities around the overlap area of successive frames, it does not cancel them. As a result, special care has to be taken to avoid introducing avoidable discontinuities in modified STFTs (see 5.1.4).

vided the down- and upsampling ratios are set to M rather than N , and in which filters $H(z)$ and $G(z)$ now also account for the effect of the non-rectangular weighting window.

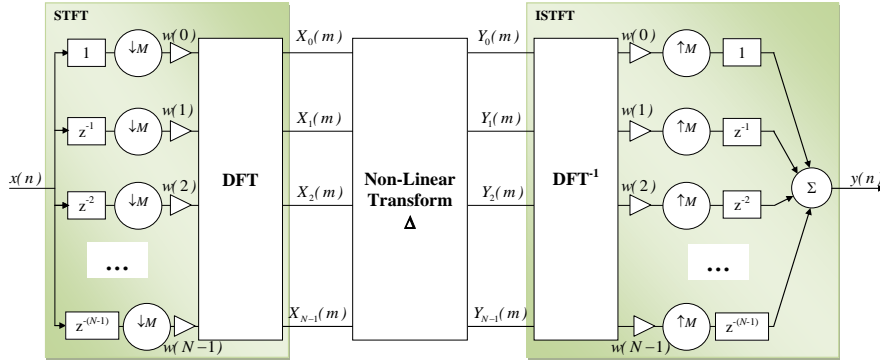


Fig. 5.7 STFT-based signal processing: block processing view

5.1.3 Perfect reconstruction

The results obtained with an STFT-based signal processing system obviously depend on the weighting window used, and on the shift between successive analysis windows.

Assuming the analysis and synthesis windows are identical, and making the further assumption that DFTs are not modified, WOLA's output is given by:

$$y(n) = \sum_{m=-\infty}^{\infty} x(n)w^2(n-mM) \quad (5.6)$$

Perfect reconstruction is therefore achieved when⁴:

$$\sum_{m=-\infty}^{\infty} w^2(n-mM) = 1 \quad \forall n \in \mathbb{Z} \quad (5.7)$$

⁴ Notice the similarity between this condition and (3.18), in which only $m=0$ and 1 are taken into account, since in the Modulated Lapped Transform the frame shift M is set to half the frame length N .

This condition is sometimes called the *Constant OverLap Add* (COLA) constraint, applied here on $w^2(n)$. It depends on the shape of the weighting window $w(n)$ and on the ratio of the frame shift M to the frame length N .

A simple example is that of the square root of the triangular (or *Bartlett*) window with $M=N/2$. The Hanning (or *Hann*) window with $M=N/4$ is usually preferred (Fig. 5.8), as its Fourier transform exhibits a better compromise between the level of its side lobes and the width of its pass-band. The square root of the Hanning window can also be used with $M=N/2$ ⁵.

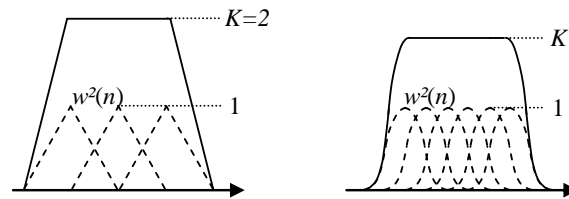


Fig. 5.8 Overlap-adding triangular windows with $M=N/2$ (left) and squared Hanning windows $M=N/4$ (right)

Usual weighting windows, when used in the COLA conditions, sum up to $K \neq 1$ (Fig. 5.8). Using the Poisson Summation Formula, it is easy to show that the value of K is given by (Smith 2007b):

$$K = \frac{\sum_{n=0}^{N-1} w^2(n)}{M} \quad (5.8)$$

It is then easy to multiply the weighting window by $1/\sqrt{K}$ to enforce (5.7).

5.1.4 Time scale modification with the phase vocoder

The phase vocoder is a specific implementation of an STFT-based processing system in which:

1. The input signal is assumed to be composed of a sum of (not necessarily harmonic) sinusoidal terms, named *partials*:

⁵ The square root of the Hanning window is nothing else than the sinusoidal window of the Modulated Lapped Transform (MLT) used in perceptual audio coders (3.1.1), with a frame shift set to half the frame-length.

$$x(n) = \sum_{i=1}^P A_i \cos(n\varphi_i + \phi_i) \quad (5.9)$$

where P is the number of partials, A_i and ϕ_i are their amplitudes and initial phases, and φ_i are their normalized angular frequencies ($\varphi_i = \omega_i/F_s$ [rad/sample]). The value of $n\varphi_i + \phi_i$ gives the *instantaneous phase* of the i th partial at sample n .

2. The output of each DFT bin is assumed to be mainly influenced by a single partial. As already mentioned in 5.1.1, satisfying this condition implies to choose long input frames, since the width of the main spectral lobe of the analysis window depends on the number N of samples in the frame.

When these assumptions are verified⁶, it is easy to show that the output of each DFT channel $X_k(m)$ ($k=0, \dots, N-1$) is a complex exponential function over m . For simplicity, let us assume that the input signal is a single imaginary exponential:

$$x(n) = Ae^{jn\varphi + \phi} \quad (5.10)$$

Then $X_k(m)$ is given by:

$$\begin{aligned} X_k(m) &= \sum_{n=0}^{N-1} Ae^{j(n+mM)\varphi + \phi} w(n) e^{-jn\Omega_k} \\ &= \left(A \sum_{n=0}^{N-1} e^{jn(\varphi - \Omega_k)} w(n) \right) e^{jmM\varphi + \phi} \quad (k=0, 1, \dots, N-1) \\ &= H_k(\varphi) e^{jmM\varphi + \phi} \end{aligned} \quad (5.11)$$

where $H_k(\varphi)$ is the frequency response of the k^{th} analysis sub-band filter for frequency φ and therefore does not depend on m . The output of each DFT channel is thus an imaginary exponential function whose angular frequency $M\varphi$ only depends on the frequency φ of the underlying partial (and *not* on the central frequency Ω_k of the DFT bin). As a result, it is predictable from any of its past values:

$$|X_k(m+1)| = |X_k(m)| \quad (5.12)$$

⁶ Clearly, these conditions are antagonistic. Since the partials of an audio signal are generally modulated in amplitude and frequency, condition 1 is only valid for short frames, while verifying condition 2 requires long input frames.

$$\text{and } \angle X_k(m+1) = \angle X_k(m) + M\varphi$$

where $\angle X_k(m)$ is the *instantaneous phase* of the output of the k th DFT channel (Fig. 5.10). Notice that only a *wrapped* version of this phase is available in practice, since it is computed as an arctangent and therefore obtained in the range $[-\pi, \pi]$.

Time-scale modification, with horizontal phase locking

Modifying the duration of the input signal can then be achieved by setting an analysis frame shift M_a in (5.5) lower (time expansion) or higher (time compression) than the synthesis frame shift M_s ⁷, and adapting the DFT channel values $Y_k(m)$ accordingly:

$$|Y_k(m)| = |X_k(m)| \quad (5.13)$$

$$\text{and } \angle Y_k(m+1) = \angle Y_k(m) + M_s \varphi_k(m)$$

in which $\varphi_k(m)$ is the frequency of the partial which mainly influences $X_k(m)$.

The initial synthesis phases $\angle Y_k(0)$ are simply set to the corresponding analysis phases $\angle X_k(0)$, and the subsequent values are obtained from (5.13). This operation is termed as *horizontal phase locking* (Fig. 5.9), as it ensures phase coherence across frames in each DFT channel independently of its neighbors. In other words, it guarantees that the short-time synthesis frames $y_m(n)$ in (5.5) will overlap coherently⁸.

Obtaining $\varphi_k(m)$ from (5.12), though, is not straightforward. Since the values of $\angle X_k(m)$ and $\angle X_k(m+1)$ are only known modulo 2π , we have:

$$\angle X_k(m+1) - \angle X_k(m) = M\varphi_k(m) + l2\pi \quad (5.14)$$

where l is an a priori unknown integer constant. Let us rewrite (5.14) as:

$$\angle X_k(m+1) - \angle X_k(m) = M(\theta_k(m) + \Omega_k) + l2\pi \quad (5.15)$$

where $\theta_k(m)$ is the frequency offset between $\varphi_k(m)$ and Ω_k (Fig. 5.10).

⁷ The synthesis frame shift M_s is left constant, rather than the analysis frame shift M_a , so as to take the COLA constraint into account.

⁸ Remember that this is only valid, though, for constant frequency partials.

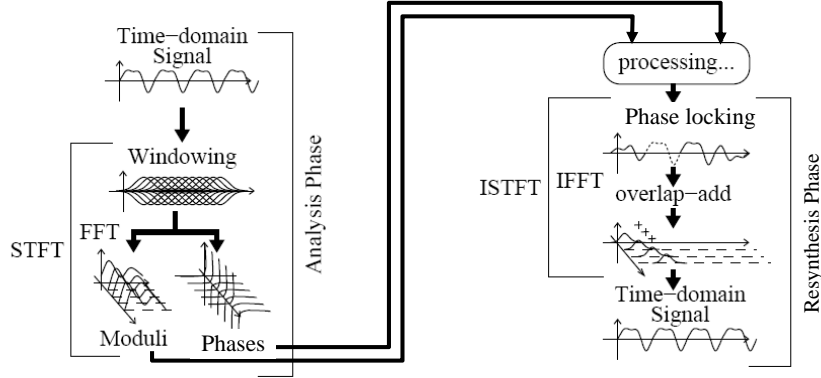


Fig. 5.9 The phase vocoder (after Arfib *et al.* 2002)

If we assume that $\theta_k(m)$ is small enough for $|M\theta_k(m)|$ to be less than π , and if $[\cdot]_{2\pi}$ denotes the reduction of the phase to its principal value in $[-\pi, \pi]$, then:

$$\begin{aligned} [M\theta_k(m)]_{2\pi} &= [\angle X_k(m+1) - \angle X_k(m) - M\Omega_k - l2\pi]_{2\pi} \\ M\theta_k(m) &= [\angle X_k(m+1) - \angle X_k(m) - M\Omega_k]_{2\pi} \end{aligned} \quad (5.16)$$

which gives the expected value of $\theta_k(m)$.

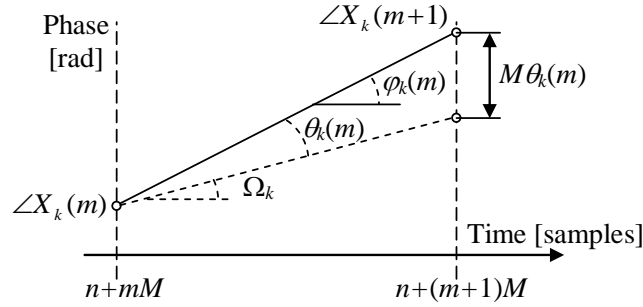


Fig. 5.10 Theoretical phase increment from frame m to frame $m+1$ in DFT channel k due to a main partial with frequency $\varphi_k(m)$ in its band-pass. In practice, $X_k(m)$ and $X_k(m+1)$ are only known modulo 2π (this is not shown here).

The assumption on the small value of $\theta_k(m)$ can be translated into an offset b in DFT bins whose central frequency is close enough to $\varphi_k(m)$ for (5.16) to be applicable:

$$\begin{aligned} |M \left(b \frac{2\pi}{N} \right)| &< \pi \\ |b| &< \frac{N}{2M} \end{aligned} \quad (5.17)$$

Since the half main spectral lobe width of a Hanning window covers approximately 2 bins, we conclude that, if a Hanning-based vocoder is designed in such a way that horizontal phase locking should be applied to the whole main lobe around each partial, then the frame shift M must be such that $M < N/4$. This condition happens to be identical as the one derived in 5.1.2.

Time-scale modification, with horizontal and vertical phase locking

The two most prominent time-scaling artifacts of the classical phase vocoder expose above are *transient smearing* and *phasiness*. Both occur even with modification factors close to one. Transient smearing is heard as a slight loss of percussiveness in the signal. Piano signals, for instance, may be perceived as having less "bite". Phasiness is heard as a characteristic coloration of the signal; in particular, time-expanded speech often sounds as if the speaker is much further from the microphone than in the original recording. Phasiness can be minimized by not only ensuring phase consistency *within* each DFT channel over time, but also as we shall see by ensuring phase coherency across the channels in a given synthesis frame. A system implementing both horizontal and vertical phase locking is called a *phase-locked vocoder*.

Coming back to the simple case of a single imaginary exponential, and further assuming that the weighting window is rectangular, $H_k(\varphi)$ in (5.11) becomes:

$$\begin{aligned} H_k(\varphi) &= \sum_{n=0}^{N-1} e^{jn(\varphi - \Omega_k)} = \frac{1 - e^{jN(\varphi - \Omega_k)}}{1 - e^{j(\varphi - \Omega_k)}} \\ &= e^{j\frac{N-1}{2}(\varphi - \Omega_k)} \left(\frac{\sin \frac{N(\varphi - \Omega_k)}{2}}{\sin \frac{(\varphi - \Omega_k)}{2}} \right) \quad (k=0, 1, \dots, N-1) \end{aligned} \quad (5.18)$$

Since the bracketed expression in (5.18) is positive in its main lobe, the following cross-channel phase relationship exists in the main lobe:

$$\begin{aligned} H_{k+1}(\varphi) - H_k(\varphi) &= e^{j\frac{N-1}{2}(\Omega_k - \Omega_{k+1})} \\ &= e^{-j\frac{N-1}{N}\pi} \end{aligned} \quad (5.19)$$

This relationship is even simplified if the weighted samples are first circularly shifted by $(N-1)/2$ samples⁹ before the DFT, as this introduces an additional linear phase factor which cancels the influence of Ω_k in $H_k(\varphi)$. As a result, (5.19) becomes:

$$H_{k+1}(\varphi) - H_k(\varphi) = 0 \quad (5.20)$$

This result can easily be generalized to any weighting window.

Puckette (1995) proposed a method called *loose phase locking*, in which the phase in each synthesis channel is implicitly influenced by the phases of the surrounding synthesis channels. The method first computes the synthesis phases in all channels $\angle Y_k(m+1)$ according to (5.13), and then modifies the results as:

$$\angle Y'_k(m+1) = \angle (Y_k(m-1) + Y_k(m) + Y_k(m+1)) \quad (k=0, 1, \dots, N-1) \quad (5.21)$$

As a result, if a DFT channel is the maximum of the DFT magnitudes, its phase is basically unchanged, since $Y_k(m-1)$ and $Y_k(m+1)$ have a much lower amplitude. Conversely, the phase of a channel surrounding the maximum will roughly be set to that of the maximum: the channels around the peaks of the DFT are roughly *phase locked*.

The fundamental limitation (and also the attraction) of the loose phase locking scheme is that it avoids any explicit determination of the signal structure: the same calculation is performed in every channel, independently of its content. Laroche and Dolson (1999) have proposed an improved phase locking scheme, named *rigid phase locking* and based on the explicit identification of peaks in the spectrum. Their new phase-updating technique starts with a coarse peak-picking stage where vocoder channels are searched for local maxima. In the simplest implementation, a channel whose amplitude is larger than its four nearest neighbors is said to be a peak. The series of peaks divides the frequency axis into *regions of in-*

⁹ In practice, since FFTs are used for implementing the phase vocoder, N is even, and samples are shifted by $N/2$. Consequently, (5.20) is only approximately verified.

fluence located around each peak. The basic idea is then to update the phases for the peak channels only, according to (5.13): the phases of the remaining channels within each region are then locked in some way to the phase of the peak channel. In particular, in their so-called *identity phase locking* scheme, Laroche and Dolson constrain the synthesis phases around each peak to be related in the same way as the analyses phases. If p is the index of dominant peak channel, the phases of all channels k ($k \neq p$) in the peak's region of influence are set to:

$$\angle Y_k(m) = \angle Y_p(m) + \angle X_k(m) - \angle X_p(m) \quad (k=0, 1, \dots, N-1) \quad (5.22)$$

Another improvement is also proposed by the authors in the *scaled phase locking* scheme, which accounts for the fact that peak channels may evolve with time.

5.1.5 Pitch shifting with the phase vocoder

The resampling-based approach

The standard pitch-scale modification technique combines time-scale modification and resampling. Assuming a pitch-scale modification by a factor β is desired (i.e., all frequencies must be multiplied by β), the first stage consists of using the phase-vocoder to perform a factor time-scale modification of the signal (its duration is multiplied by β). In the second stage, the resulting signal is resampled at a new sampling frequency F_s/β , where F_s is the original sampling period. The output signal ends up with the same duration as the original signal, but its frequency content has been expanded by a factor β during the resampling stage, which is the desired result (Fig. 5.11).

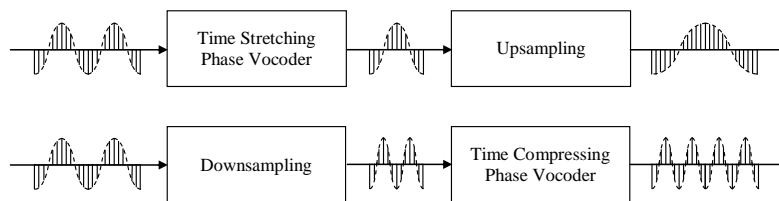


Fig. 5.11 A possible implementation of a pitch-shifter using a phase vocoder. Top: downward pitch-scaling ($\beta < 1$); Bottom: upward pitch scaling.

Note that it is possible to reverse the order of these two stages, which yields the same result if the window size is multiplied by β in the phase-

vocoder time-scaling stage. However, the cost of the algorithm is a function of the modification factor and of the order in which the two stages are performed. For example, for upward pitch-shifting ($\beta > 1$), it is more advantageous to resample first and then time-scale, because the resampling stage yields a shorter signal. For downward pitch-shifting, it is better to time-scale first and then resample, because the time-scaling stage yields a signal of smaller duration (as shown on Fig. 5.11).

This standard technique has several drawbacks. Its computational cost is a function of the modification factor. If the order in which the two stages are performed is fixed, the cost becomes increasingly large for larger upward or downward modifications. An algorithm with a fixed cost is usually preferable. Another drawback of the standard technique is that only *one* "linear" pitch-scale modification is allowed i.e., the frequencies of all the components are multiplied by the same factor. As a result, harmonizing a signal (i.e., adding several copies pitch-shifted with different factors) requires repeated processing at a prohibitive cost for real-time applications.

The STFT-based approach

A more flexible algorithm has been proposed by Laroche and Dolson (1999b), which allows non-linear frequency modifications, enabling the same kind of alterations that usually require non real-time sinusoidal analysis/synthesis techniques.

The underlying idea behind the new techniques consists of identifying peaks in the short-term Fourier transform, and then translating them to new arbitrary frequencies. If the relative amplitudes and phases of the bins around a sinusoidal peak are preserved during the translation, then the time-domain signal corresponding to the shifted peak is simply a sinusoid at a different frequency, modulated by the same analysis window.

For instance, in the case of a single imaginary exponential signal $x(n) = Ae^{jn\varphi + \phi}$, input frames will be of the form:

$$x_m(n) = A w(n + mM) e^{j(n+mM)\varphi + \phi} \quad (5.23)$$

and (5.11) shows that $X_k(m)$ only depends on k via $(\varphi - \Omega_k)$. As a result, imposing¹⁰:

$$Y_k(m) = X_{(k+S) \bmod N}^{(*)}(m) \quad (k = 0, 1, \dots, N-1) \quad (5.24)$$

¹⁰ Notice the "mod N " in (5.24), which makes sure that, if a shifted region of influence spills out of the frequency range $[0, 2\pi]$, it is simply reflected back into $[0, 2\pi]$ after complex conjugation (indicated by our non standard use of the "(*)" notation) to account for the fact that the original signal is real.

in which S is an integer number of DFT bins, will result in output frames of the form:

$$y_m(n) = A w(n + mM) e^{j(n+mM)(\varphi+\Delta\varphi)+\phi} \quad (5.25)$$

with $\Delta\varphi = S \frac{2\pi}{N}$. These output frames, however, will not overlap-add coherently into:

$$y(n) = A e^{jn(\varphi+S\frac{2\pi}{N})+\phi} \quad (5.26)$$

since (5.24) does not ensure that the peak phases are consistent from one frame to the next. As a matter of fact, since the frequency has been changed from φ to $\varphi+\Delta\varphi$, phase coherency requires rotating all phases by $M\Delta\varphi$ and accumulating this rotation for each new frame, which results in the final pitch-scaling scheme:

$$Y_k(m) = X_{(k+S)\bmod N}(m) e^{jmM\Delta\varphi} \quad (k = 0, 1, \dots, N-1) \quad (5.27)$$

Pitch-shifting a more complex signal is achieved by first detecting spectral peaks and regions of influence, as in the phase-locked vocoder, and then shifting each region of influence separately by β , using (5.27) with $\Delta\varphi=(\beta-1)\varphi$. Shifted areas of influence that overlap in frequency are simply added together.

When the value of the frequency shift $\Delta\varphi$ does not correspond to an integer number of bins S (which is the most general case), it can either be rounded to the closest integer value of S (which is very acceptable for large DFT sizes and low sampling rates, for which DFT channel are very narrow), or linear interpolation can be used to distribute each shifted bin into existing DFT bins (see Laroche and Dolson 1999b).

It is useful to note that because the channels around a given peak are rotated by the same angle, the differences between the phases of the channels around a peak in the input STFT are preserved in the output STFT short-term Fourier transform. This is similar to the "Identity Phase-Locking" scheme, which dramatically minimizes the phasiness artifact often encountered in phase-vocoder time or pitch-scale modifications.

Notice also, finally, that the exact value of the frequency φ of each partial is not required, since only $\Delta\varphi$ is used in (5.27). This is an important savings compared to the standard phase vocoder, and makes this approach more robust.

5.2 MATLAB proof of concept: ASP_audio_effects.m

In this section, we show that STFT-based signal processing provides very flexible tools for audio signal processing. We start by radically modifying the phases of the STFT, to create a robotization effect (5.2.1). Then we examine the MATLAB implementation of a phase vocoder, with and without vertical phase locking (5.2.2). We conclude the Chapter with two pitch-scale modification algorithms, which use the phase vocoder in various aspects (5.2.3).

5.2.1 STFT-based audio signal processing

We first implement a basic STFT-based processing scheme, using the same weighting window for both analysis and synthesis, and with very simple processing of the intermediate DFTs.

Weighting windows

Since weighting windows play an important part in STFT-based signal processing, it is interesting to check their features first. MATLAB proposes a handy tool for that: the `wvtool` function. Let us use it for comparing the rectangular, Hanning, and Blackman windows. Clearly, the spectral leakage of the Hanning and Blackman windows are lower than those of the rectangular and `sqrt(Hanning)` windows. By zooming on the the main spectral lobes, one can check that this is compensated by a higher lobe width (Fig. 5.12): the spectral width of the rectangular window is half that of the Hanning window, and a third of that of the Blackman window.

```
N=100;  
wvtool(boxcar(N),hanning(N),blackman(N));
```

The Constant OverLap-Add (COLA) constraint

Let us now examine the operations involved in weighted overlap-add (WOLA). When no modification of the analysis frames is performed, one can to the least expect the output signal to be very close to the input signal. In particular, a constant input signal should result in a constant output signal. This condition is termed as Constant OverLap-Add (COLA) constraint. It is only strictly met for specific choices of the weighting window and of the window shift, as we shall see. We now check this on a chirp signal.

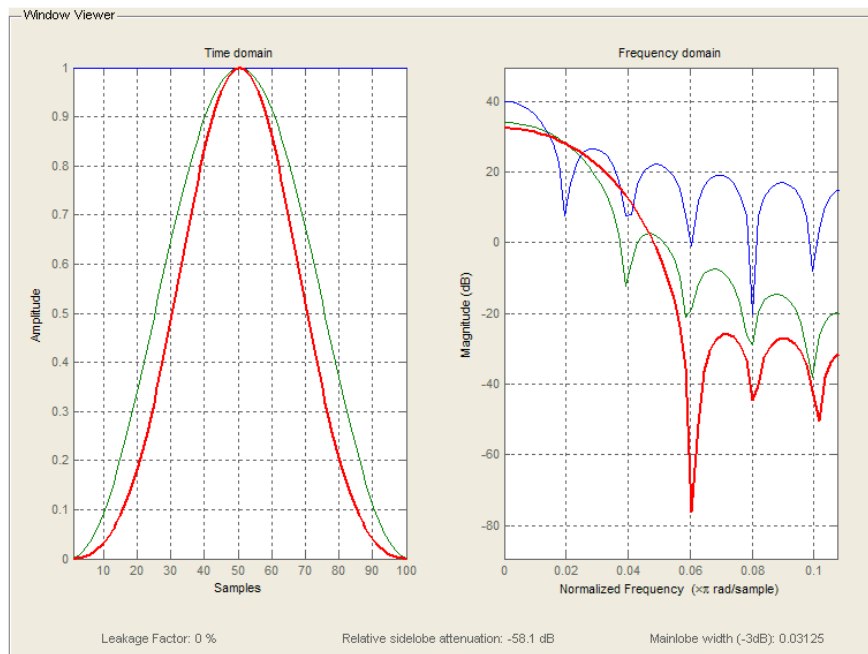


Fig. 5.12 The 100-samples rectangular, Hanning, and Blackman weighting windows (left) and their main spectral properties (right)

```

Fs=8000;
input_signal=chirp((0:Fs)/Fs,0,1,2000)';
specgram(input_signal,1024,Fs,256);
    
```

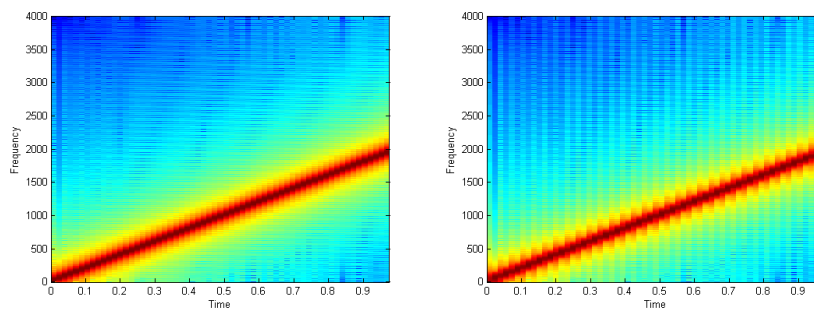


Fig. 5.13 Spectrogram of a chirp. Left: original signal; Right: after weighted overlap-add using a Hanning window and a frame shift set to half the frame length

We actually use the periodic version of the Hanning window, which better meets the COLA constraint for various values of the frame shift than the default symmetric version. It is easy to plot the SNR of the analysis-synthesis process, for a variable frame shift between 1 and 512 samples and a frame length N of 512 samples (Fig. 5.14, left). The resulting SNR is indeed very high for values of the frame shift M equal to $N/4$, $N/8$, $N/16$, ..., 1. These values are said to meet the COLA constraint for the squared Hanning window. In practice, values of the frame shift that do not meet the COLA constraint can still be used, provided the related SNR is higher than the local signal-to-mask ratio (SMR) due to psycho-acoustic effects (see Chapter 3). Using the highest possible value for the frame shift while still reaching a very high SNR minimizes the computational load of the system.

```
COLA_check=zeros(length(input_signal),2);
frame_length=512;
for frame_shift=1:frame_length
    % Amplitude normalization for imposing unity COLA
    window=hanning (frame_length,'periodic');
    COLA_ratio=sum(window.*window)/frame_shift;
    window=window/sqrt(COLA_ratio);

    output_signal=zeros(length(input_signal),1);
    pin=0;    % current position in the input signal
    pout=0;   % current position in the output signal
    while pin+frame_length<length(input_signal)
        % Creating analysis frames
        analysis_frame = ...
            input_signal(pin+1:pin+frame_length).* window;

        % Leaving analysis frames untouched
        synthesis_frame=analysis_frame;

        % Weighted OverLap-Add (WOLA)
        output_signal(pout+1:pout+frame_length) = ...
            output_signal(pout+1:pout+frame_length)+...
            synthesis_frame.*window;

        % Checking COLA for two values of the frame shift
        if (frame_shift==frame_length/2)
            COLA_check(pout+1:pout+frame_length,1)=...
                COLA_check(pout+1:pout+frame_length,1)...
                +window.*window;
        elseif (frame_shift==frame_length/4)
            COLA_check(pout+1:pout+frame_length,2)=...
                COLA_check(pout+1:pout+frame_length,2)...
                +window.*window;
        end;

        pin=pin+frame_shift;
        pout=pout+frame_shift;
    end;
```

```

% Storing the output signal for frame shift = frame_length/2
if (frame_shift==frame_length/2)
    output_half=output_signal;
end;

% Using the homemade snr function introduced in Chapter 2,
% and dropping the first and last frames, which are only
% partially overlap-added.
snr_values(frame_shift)=snr(...
    input_signal(frame_length+1:end-frame_length),...
    output_signal(frame_length+1:end-frame_length),0);

end

plot((1:frame_length)/frame_length,snr_values);

```

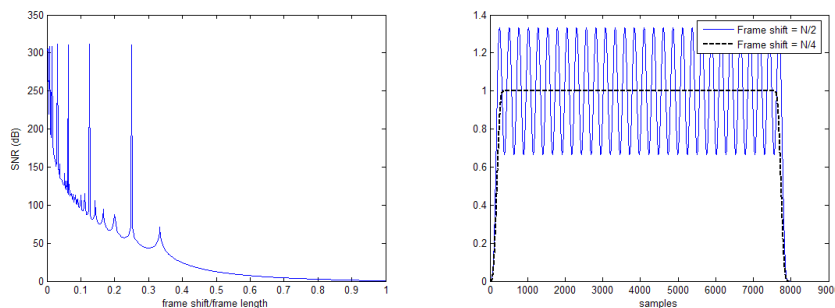


Fig. 5.14 Examining the COLA constraint for a Hanning window. Left: SNR for various values of the ratio frame shift/frame length; Right: result of the OLA with $M=N/2$ (non COLA) and $N/4$ (COLA)

When the input signal is a unity constant, the output signal can indeed be close to unity... or not, depending the value of the frame shift. For the Hanning window, obviously, a frame shift of $N/4$ meets the COLA constraint, while $N/2$ leads to an important amplitude ripple (Fig. 5.14, right). These values also depend on the weighting window.¹¹

```

plot(COLA_check(:,1)); hold on;
plot(COLA_check(:,2),'k--','linewidth',2); hold off;

```

A bad choice of the frame shift may lead to important perceptual degradation of the output audio signal, as confirmed by listening to the output produced for $M=N/2$ (see Fig. 5.13, right for the related spectrogram).

```

soundsc(output_half, Fs);

```

¹¹ See Appendix 1 in the `audio_effects.m` file for a test on the square-root Hanning window.

```
specgram(output_half,1024,Fs,256);
```

STFT-based signal processing

We now add an STFT/ISTFT step. The choice of the frame length N , of the type of weighting window and of the frame shift, depends on the type of processing applied to frequency bands. For frequency-selective processing, a large value of N is required, as the bandwidth of the DFT channels is inversely proportional to N . In this Section, we will implement a simple robotization effect on a speech signal, which is not a frequency-selective modification. We therefore set N to 256 samples (but many other values would match our goal). We choose a Hanning window, with a frame shift of 80 samples. This value does not meet the COLA constraint for the Hanning window, but the effect we will apply here does not attempt to maintain the integrity of the input signal anyway.

Let us process the **speech.wav** file, which contains the sentence *Paint the circuits* sampled at 8 kHz. We first plot its spectrogram using the same frame length and frame shift as that of our STFT. The resulting plot is not especially pretty, but it shows exactly the data that we will process. The spectrogram reveals the harmonic structure of the signal, and shows that its pitch is a function of time (Fig. 5.15, left).

```
frame_length=256;
frame_shift=80;
window=hanning (frame_length,'periodic');
COLA_ratio=sum(window.*window)/frame_shift;
window=window/sqrt(COLA_ratio);

[input_signal,Fs]=wavread('speech.wav');
specgram(input_signal,frame_length,Fs,window);
```

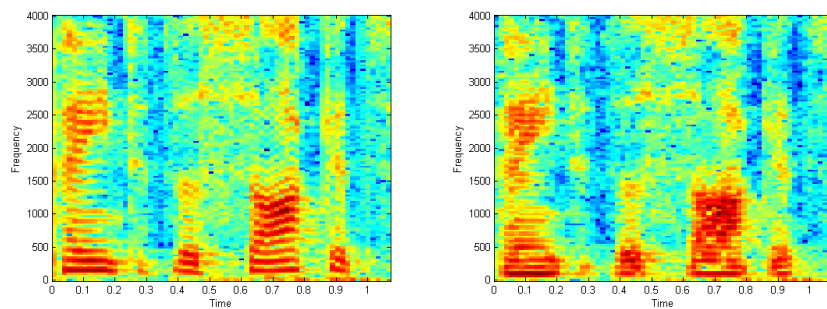


Fig. 5.15 Spectrogram of the sentence *Paint the circuits*. Left: original signal, as produced by the STFT; Right: robotized signal

When no processing of the intermediate DFTs is performed, the output signal is thus equal to the input signal. It is easy to modify the amplitudes or the phases of the STFT, to produce audio effects. Let us test a simple robotization effect, for instance, by setting all phases to zero. This produces a disruptive perceptual effect, at the periodicity of the frame shift (Fig. 5.16). When the shift is chosen small enough, and the effect is applied to speech, it is perceived as artificial constant pitch. This simple effect appears on the spectrogram as a horizontal reshaping of the harmonics (Fig. 5.15, right). It reveals the importance of phases modifications (and more specifically of phase discontinuities) in the perception of a sound.

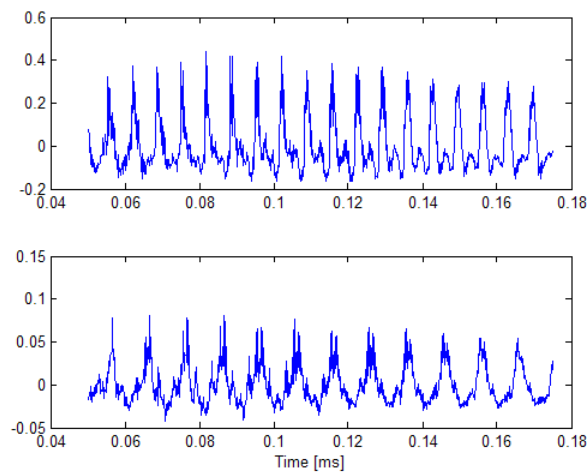


Fig. 5.16 Zoom on the first vowel of *Paint the circuits*. Top: original signal; Bottom: robotized signal with frame shift set to 10 ms (hence producing an artificial pitch of 100 Hz)

```
output_signal=zeros(length(input_signal),1);
pin=0;pout=0;
while pin+frame_length<length(input_signal)
    % STFT
    analysis_frame=input_signal(pin+1:pin+frame_length).* window;
    dft=fft(analysis_frame);

    % Setting all phases to zero
    dft=abs(dft);

    % ISTFT
    synthesis_frame=ifft(dft).*window;
    output_signal(pout+1:pout+frame_length) = ...
        output_signal(pout+1:pout+frame_length)+synthesis_frame;
```

```

        pin=pin+frame_shift;
        pout=pout+frame_shift;
    end;

    % zoom on a few ms of signal, before and after robotization
    clf;
    range=(400:1400);
    subplot(211);
    plot(range/Fs,input_signal(range))
    subplot(212);
    plot(range/Fs,output_signal(range))

    specgram(output_signal,frame_length,Fs>window);

```

5.2.2 Time scale modification

In this Section we examine methods for modifying the duration of an input signal without changing its audio spectral characteristics (e.g., without changing the pitch of any of the instruments). The test sentence we use here contains several chunks, sampled at 44100 Hz (Fig. 5.17, left). It starts with 1000 Hz sine wave, followed by a speech excerpt. The sound of a (single) violin comes next, followed by an excerpt of a more complex polyphonic musical piece (*Time* by Pink Floyd).

```

[input_signal,Fs]=wavread('time_scaling.wav');

frame_length=2048;
frame_shift=frame_length/4;
window=hanning (frame_length,'periodic');
COLA_ratio=sum(window.*window)/frame_shift;
window=window/sqrt(COLA_ratio);

specgram(input_signal,frame_length,Fs>window);

```

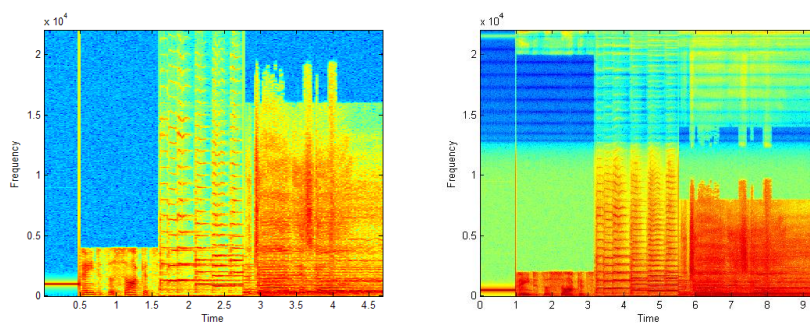


Fig. 5.17 Spectrogram of the test signal. Left: original signal; Right: after time-stretching by a factor of two

Interpolating the signal

Increasing the length of the input signal by a factor of two, for instance, is easily obtained by interpolating the signal, using the `resample` function provided by MATLAB, and not telling the digital-to-analog converter (DAC) about the sampling frequency change. This operation, however, also changes the frequency content of the signal: all frequencies are divided by two. Similarly, speeding-up the signal by a factor two would multiply all frequencies by two. This is sometimes referred to as the "chipmunk effect". Notice incidentally the aliasing introduced by the imperfect low-pass filter used by `resample` (Fig. 5.17, right).

```
resampled_signal=resample(input_signal,2,1);
specgram(resampled_signal,frame_length,Fs>window);
```

Time-scale modification with Weighted OverLap-Add (WOLA)

It is also possible to modify the time scale by decomposing it into overlapping frames, changing the analysis frame shift into a different synthesis frame shift, and applying weighted overlap-add (WOLA) to the resulting synthesis frames. While this technique works well for unstructured signals, its application to harmonic signals produces unpleasant audio artifacts at the frequency of the synthesis frame rate, due to the loss of synchronicity between excerpts of the same partials in overlapping synthesis frames (Fig. 5.18). This somehow reminds us of the robotization effect obtained in Section 1.3.

```
frame_length=2048;
synthesis_frame_shift=frame_length/4;
window=hanning (frame_length,'periodic');
COLA_ratio=sum(window.*window)/synthesis_frame_shift;
window=window/sqrt(COLA_ratio);

time_scaling_ratio=2.85;
analysis_frame_shift=...
    round(synthesis_frame_shift/time_scaling_ratio);

pin=0;pout=0;
output_signal=zeros(time_scaling_ratio*length(input_signal),1);

while (pin+frame_length<length(input_signal)) ...
    && (pout+frame_length<length(output_signal))

    analysis_frame = input_signal(pin+1:pin+frame_length).*...
        window;
    synthesis_frame = analysis_frame.* window;
    output_signal(pout+1:pout+frame_length) = ...
        output_signal(pout+1:pout+frame_length)+synthesis_frame;

    pin=pin+analysis_frame_shift;
    pout=pout+synthesis_frame_shift;

    % Saving frames for later use
    if (pin==2*analysis_frame_shift) % 3rd frame
```

```

        frame_3=synthesis_frame;
    elseif (pin==3*analysis_frame_shift) % 4th frame
        frame_4=synthesis_frame;
    end;

end;

% Plot two overlapping synthesis frames and show the resulting
output signal
clf;
ax(1)=subplot(211);
range=(2*synthesis_frame_shift:2*synthesis_frame_shift+...
    frame_length-1);
plot(range/Fs,frame_3)
hold on;
range=(3*synthesis_frame_shift:3*synthesis_frame_shift+...
    frame_length-1);
plot(range/Fs,frame_4,'r')
ax(2)=subplot(212);
range=(2*synthesis_frame_shift:3*synthesis_frame_shift+...
    frame_length-1);
plot(range/Fs,output_signal(range))
xlabel('Time [ms]');
linkaxes(ax,'x');
set(gca,'xlim',[0.045 0.06]);

soundsc(output_signal,Fs);

```

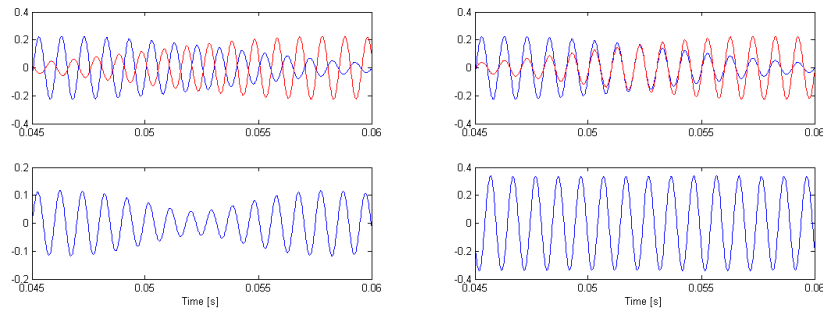


Fig. 5.18 Time-scale modification of a sinusoidal signal. Top: zoom on the overlap of two successive synthesis frames; Bottom: the resulting output signal; Left: using WOLA alone; Right: using the phase vocoder.

Time-scale modification with the phase vocoder

Let us now modify the time scale of the input signal without affecting its frequency content, by using a phase vocoder, which is a STFT-based signal processing system with specific hypotheses on the STFT. Time-scaling is again achieved by modifying the analysis frame shift without changing the synthesis frame shift, but the STFT has to be modified, so as to avoid creating phase mismatches between overlapped synthesis frames.

We use the periodic Hanning window, which provides a low spectral leakage while its main spectral lobe width is limited to $4F_s/N$. The choice of the frame length N is dictated by a maximum bandwidth constraint for the DFT sub-band filters, since the phase vocoder is based on the hypothesis that each DFT bin is mainly influenced by a single partial, i.e. a sinusoidal component with constant frequency. For a Hanning window, each sinusoidal component will drive 4 DFT bins, the width of each is F_s/N . Clearly, the higher the value of N , the better the selectivity of the sub-band filters. On the other hand, a high value of N tends to break the stationarity hypothesis on the analysis frame (i.e., the frequency of partials will no longer be constant). We set N to 2048 samples here, which imposes the width of each DFT bin to 21 Hz, and the bandwidth of the DFT sub-band filters to 86 Hz. We set the frame shift to $N/4=512$ samples, which meets the COLA constraint for the Hanning window. What is more, this choice obeys the Shannon theorem for the output of each DTF bin, seen as the output of a sub-band filter. The result is much closer to what is expected (Fig. 5.19).

```
NFFT=2048;
frame_length=NFFT;
synthesis_frame_shift=frame_length/4;
window=hanning (frame_length,'periodic');
COLA_ratio=sum(window.*window)/synthesis_frame_shift;
window=window/sqrt(COLA_ratio);

time_scaling_ratio=2.65;
analysis_frame_shift=round(synthesis_frame_shift/...
    time_scaling_ratio);
% Central frequency of each DFT channel
DFT_bin_freqs=((0:NFFT/2-1)*2*pi/NFFT)';

pin=0;pout=0;

output_signal=zeros(time_scaling_ratio*length(input_signal),1);
last_analysis_phase=zeros(NFFT/2,1);
last_synthesis_phase=zeros(NFFT/2,1);

while (pin+frame_length<length(input_signal)) ...
    && (pout+frame_length<length(output_signal))

    % STFT
    analysis_frame=input_signal(pin+1:pin+frame_length).* window;
    dft=fft(fftshift(analysis_frame));
    dft=dft(1:NFFT/2);

    % PHASE MODIFICATION
    % Find phase for each bin, compute by how much it increased
    % since last frame.
    this_analysis_phase = angle(dft);
    delta_phase = this_analysis_phase - last_analysis_phase;
    phase_increment=delta_phase-analysis_frame_shift...
        *DFT_bin_freqs;

    % Estimate the frequency of the main partial for each bin
    principal_determination=mod(phase_increment+pi,2*pi)-pi;
```

```

    partials_freq=principal_determination/analysis_frame_shift+...
        DFT_bin_freqs;

    % Update the phase in each bin
    this_synthesis_phase=last_synthesis_phase+...
        synthesis_frame_shift*partials_freq;

    % Compute DFT of the synthesis frame
    dft= abs(dft).* exp(j*this_synthesis_phase);

    % Remember phases
    last_analysis_phase=this_analysis_phase;
    last_synthesis_phase=this_synthesis_phase;

    % ISTFT
    dft(NFFT/2+2:NFFT)=fliplr(dft(2:NFFT/2)');
    synthesis_frame = fftshift(real(ifft(dft))).* window;
    output_signal(pout+1:pout+frame_length) = ...
        output_signal(pout+1:pout+frame_length)+synthesis_frame;

    pin=pin+analysis_frame_shift;
    pout=pout+synthesis_frame_shift;

    % Saving the estimated frequency of partials for later use
    if (pin==2*analysis_frame_shift) % 3rd frame
        partials_freq_3=partials_freq;
    end;

end;

specgram(output_signal(1:end-5000),frame_length,Fs>window);

```

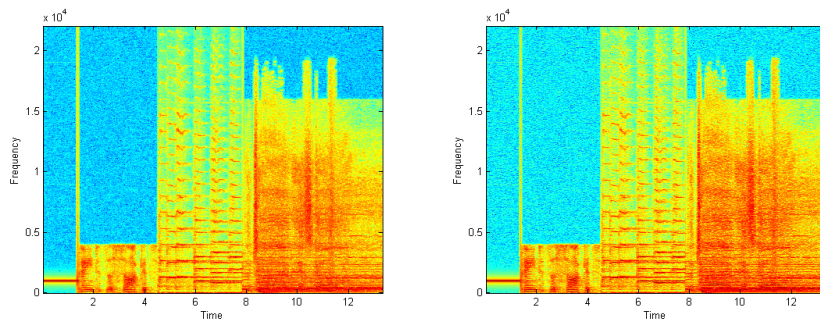


Fig. 5.19 Spectrogram of the test signal after time-stretching by a factor 2.85 using a phase vocoder with horizontal phase locking (left) and with both horizontal and vertical phase locking (right)

The phasing effect we had observed in the previous method has now disappeared (Fig. 5.18, right).

Let us check how the normalized frequency of the initial sinusoid (i.e. $1000 \cdot 2 \cdot \pi / F_s = 0.142$ rad/s) has been estimated. The first spectral line of

the sinusoid mainly influences DTF bins around index $1000/(44100/2048)=46.43$. As observed on Fig. 5.20 (left), bins 42 to 51 (which correspond to the main spectral lobe) have indeed measured the correct frequency.

```
subplot(2,1,1);
plot((0:NFFT/2-1),partials_freq_3);
set(gca,'xlim',[25 67]);

subplot(2,1,2);
third_frame=input_signal(2*analysis_frame_shift:2*analysis_frame_shift+frame_length-1);
dft=fft(third_frame.* window);
dft=20*log10(abs(dft(1:NFFT/2)));
plot((0:NFFT/2-1),dft);
set(gca,'xlim',[25 67]);
```

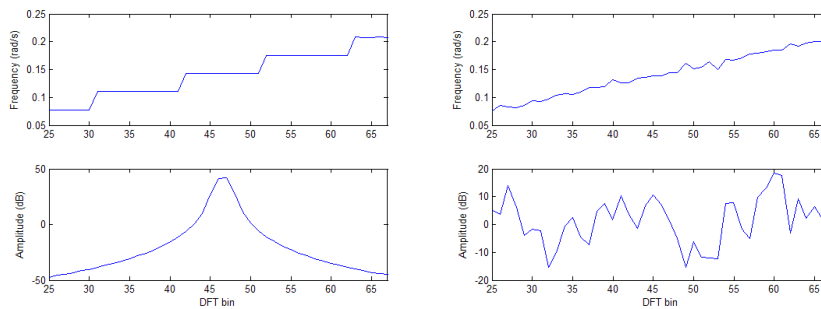


Fig. 5.20 Estimated (normalized) frequency of the partials in DFT bins 25 to 65 (top) and spectral amplitude for the same bins (bottom), in a phase vocoder. Left: in the sinusoid; Right: in more complex polyphonic music

However, for more complex sounds (as in the last frame of our test signal), the estimated partial frequency in neighboring bins vary a lot (Fig. 5.20, right).

```
subplot(2,1,1);
plot((0:NFFT/2-1),partials_freq);
set(gca,'xlim',[25 67]);

subplot(2,1,2);
dft=fft(analysis_frame);
dft=20*log10(abs(dft(1:NFFT/2)));
plot((0:NFFT/2-1),dft);
set(gca,'xlim',[25 67]);
```

As a result, although the spectrogram of the time-scaled signal looks similar to that of the original signal (except of course for the time axis), it exhibits significant phasiness and transient smearing.

The phase-locked vocoder

We have seen in the previous paragraphs that, in the case of a sinusoidal input signal, the estimated frequency of the main partial in several neighboring DFT bins (around the frequency of the sinusoid) is somehow constant. One of the reasons of the phasiness in the previous version of the phase vocoder comes from the fact that it does not enforce this effect. If other small amplitude partials are added to the sinusoid, each DFT bin computes its own partial frequency, so that these estimations in neighboring bins will only coincide by chance. The phase-locked vocoder changes this, by locking the estimation of partial frequencies in bins surrounding spectral peaks. This is called *vertical phase-locking*. In the following MATLAB lines, we give an implementation of the phase modification stage in the identity phase-locking scheme proposed in (Laroche and Dolson 1999).

MATLAB function involved:

- function [maxpeaks,minpeaks] = findpeaks(x) returns the indexes of the local maxima (maxpeaks) and of the local minima (minpeaks) in array x.

```
% PHASE MODIFICATION
% Find phase for each bin, compute by how much it increased
% since last frame.
this_analysis_phase = angle(dft);
delta_phase = this_analysis_phase - last_analysis_phase;
phase_increment=delta_phase-analysis_frame_shift*...
    DFT_bin_freqs;

% Find peaks in DFT.
peaks = findpeaks(abs(dft));

% Estimate the frequency of the main partial for each bin
principal_determination(peaks)=...
    mod(phase_increment(peaks)+pi,2*pi)-pi;
partials_freq(peaks)=principal_determination(peaks)/...
    analysis_frame_shift+DFT_bin_freqs(peaks);

% Find regions of influence around peaks
regions = round(0.5*(peaks(1:end-1)+peaks(2:end)));
regions = [1;regions;NFFT/2];

% Set the frequency of partials in regions of influence to
% that of the peak (this is not strictly needed; it is used
% for subsequent plots)
for i=1:length(peaks)
    partials_freq(regions(i):regions(i+1)) = ...
        partials_freq(peaks(i));
end

% Update the phase for each peak
this_synthesis_phase(peaks)=last_synthesis_phase(peaks)+...
```

```

synthesis_frame_shift*partials_freq(peaks);
% force identity phase locking in regions of influence
for i=1:length(peaks)
    this_synthesis_phase(regions(i):regions(i+1)) = ...
        this_synthesis_phase(peaks(i)) + ...
        this_analysis_phase(regions(i):regions(i+1))-...
        this_analysis_phase(peaks(i));
end

```

As a result of vertical phase-locking, a much larger number of bins around the main spectral line of our sinusoid have been assigned the same partial frequency. More importantly, this property has been enforced in more complex sounds, as shown in the last frame of our test signal.

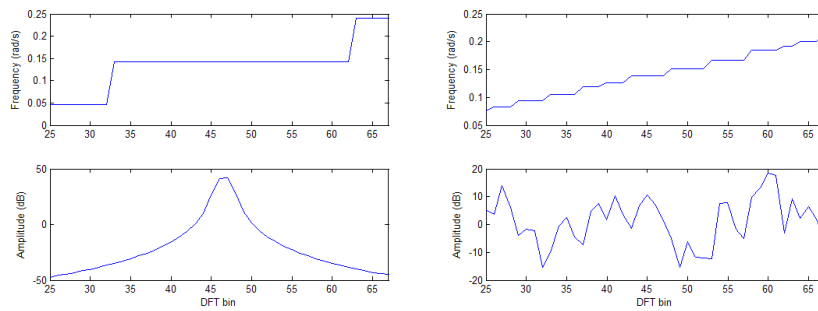


Fig. 5.21 Estimated (normalized) frequency of the partials in DFT bins 25 to 65 (top) and spectral amplitude for the same bins (bottom), in a phase-locked vocoder. Left: in the sinusoid; Right: in more complex polyphonic music

The result is improved, although it is still far from natural. It is possible to attenuate transient smearing by using separate analysis windows for stationary and transient sounds (as done in the MPEG audio coders). We do not examine this option here.

5.2.3 Pitch modification

The simplest way to produce a modification of the frequency axis of a signal is simply to resample it, as already seen in Section 2.1. This method, however, also produces time modification. In this Section we focus on methods for modifying the pitch of an input signal without changing its duration.

Time-scale modification and resampling

In order to avoid the time-scaling effect of a simple resampling-based approach, a compensatory time-scale modification can be applied, using a phase vocoder. Let us multiply the pitch by a factor 0.7, for instance. We first multiply the duration of the input signal by 0.7, and then resample it by 10/7, so as to recover the original number of samples.

MATLAB function involved:

`function [output_signal] = phase_locked_vocoder(input_signal, time_scaling_ratio)` is a simple implementation of the phase locked vocoder described in (Laroche and Dolson 1999).

```
[input_signal,Fs]=wavread('time_scaling.wav');
time_scaling_ratio=0.7;
resampling_ratio=2;

output_signal=phase_locked_vocoder(input_signal,...
    time_scaling_ratio);
pitch_modified_signal=resample(output_signal,10,7);

soundsc(input_signal,Fs);
soundsc(pitch_modified_signal,Fs);
```

The time scale of the output signal is now identical to that of the input signal. Notice again the aliasing introduced by the imperfect low-pass filter used by `resample` (Fig. 5.22, left).

```
specgram(pitch_modified_signal(1:end-5000),frame_length,...
    Fs>window);
```

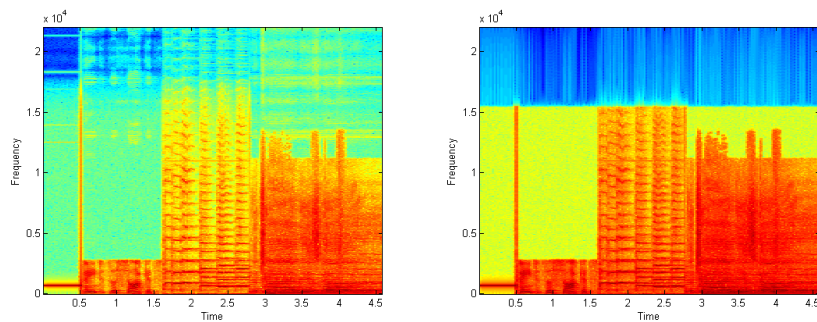


Fig. 5.22 Spectrogram of the test signal after pitch modification by a factor 0.7, using a phase vocoder followed by a resampling stage (left) and a fully integrated STFT-based approach (right)

The STFT-based approach

It is also possible, and much easier, to perform pitch modification in the frequency domain by translating partials to new frequencies, i.e. inside the phase vocoder. This technique is much more flexible, as it allows for non-linear modification of the frequency axis.

Notice that the implementation we give below is the simplest possible version: the peak frequencies are not interpolated for better precision, and the shifting is rounded to an integer number of bins. This will produce modulation artifacts on sweeping sinusoids, and phasing artifacts on audio signals such as speech. Moreover, when shifting the pitch down, peaks are truncated at DC, whereas they should be reflected with a conjugation sign.

```
NFFT=2048;
frame_length=NFFT;
frame_shift=frame_length/4;
window=hanning (frame_length,'periodic');
COLA_ratio=sum(window.*window)/frame_shift;
window=window/sqrt(COLA_ratio);

pitch_scaling_ratio=0.7;

% Central frequency of each DFT channel
DFT_bin_freqs=((0:NFFT/2-1)*2*pi/NFFT)';

pin=0;pout=0;

output_signal=zeros(length(input_signal),1);
accumulated_rotation_angles=zeros(1,NFFT/2);

while pin+frame_length<length(input_signal)

    % STFT
    analysis_frame = input_signal(pin+1:pin+frame_length)...
        .* window;
    dft=fft(fftshift(analysis_frame));
    dft=dft(1:NFFT/2);

    % Find peaks in DFT.
    peaks = findpeaks(abs(dft));

    % Find regions of influence around peaks
    regions = round(0.5*(peaks(1:end-1)+peaks(2:end)));
    regions = [1;regions;NFFT/2];

    % Move each peak in frequency according to modification
    % factor.

    modified_dft = zeros(size(dft));
    for u=1:length(peaks)

        % Locate old and new bin.
        old_bin = peaks(u)-1;
        new_bin = round(pitch_scaling_ratio*old_bin);

        % Be sure to stay within 0-NFFT/2 when
        % shifting/copying the peak bins
        if(new_bin-old_bin+regions(u) >= NFFT/2) break; end;
        if(new_bin-old_bin+regions(u+1) >= NFFT/2)
            regions(u+1) = NFFT/2 - new_bin + old_bin;
```

```

    end;
    if(new_bin-old_bin+regions(u) <= 0)
        regions(u) = 1 - new_bin + old_bin;
    end;

    % Compute the rotation angle required, which has
    % to be cumulated from frame to frame
    rotation_angles=accumulated_rotation_angles(old_bin+1)...
        + 2*pi*frame_shift*(new_bin-old_bin)/NFFT;

    % Overlap/add the bins around the peak, changing the
    % phases accordingly
    modified_dft(new_bin-old_bin+(regions(u):regions(u+1))) = ...
        modified_dft(new_bin-old_bin+(regions(u):regions(u+1)))...
        + dft(regions(u):regions(u+1)) * exp(j*rotation_angles);
    accumulated_rotation_angles((regions(u):regions(u+1))) = ...
        rotation_angles;

end

% ISTFT
modified_dft(NFFT/2+2:NFFT)=fliplr(modified_dft(2:NFFT/2)');
synthesis_frame = fftshift(real(ifft(modified_dft))).* window;
output_signal(pout+1:pout+frame_length) = ...
    output_signal(pout+1:pout+frame_length)+synthesis_frame;

pin=pin+frame_shift;
pout=pout+frame_shift;

end;

```

The time scale of the output signal is still identical to that of the input signal, and the frequency content above $F_s \cdot \text{time_scaling_ratio}$ is set to zero (Fig. 5.22, right).

```
specgram(output_signal(1:end-5000), frame_length, Fs, window);
```

5.3 Going further

For people interested in the use of transforms for audio processing, (Sethares 2007) is an excellent reference.

An important research area for phase vocoders is that of transient detection, for avoiding the transient smearing effect mentioned in this Chapter (see Röbel 2003 for instance).

One of the extensions of phase vocoders which have not been dealt with in this Chapter is the fact that not only phases but also amplitudes should be modified when performing time-scale modifications. In the frequency domain, a chirp sinusoid has a wider spectral lobe than a constant-frequency sinusoid. Exploiting this fact, however, would imply an important increase in computational and algorithmic complexity. Sinusoidal modeling techniques (McAulay and Quatieri 1986), which have been de-

veloped in parallel to the phase vocoder, are more suited to handle such effects.

For monophonic signals, such as speech, a number of time-domain techniques known as synchronized OLA (SOLA) have also been developed with great success for time scaling and pitch modification, given their very low computational cost compared to the phase vocoder. The waveform Similarity SOLA (WSOLA), and pitch-synchronous (PSOLA) techniques are presented in a unified framework in Verhelst *et al.* (2000). Their limitations are exposed in (Laroche 1998).

The multi-band resynthesis OLA (MBROLA) technique (Dutoit 1993) attempts to take the best of both worlds: it performs off-line frequency-domain modifications of the signal to make it more amenable to time-domain time scaling and pitch modification. Recently, Laroche (2003) has also proposed a vocoder-based technique for monophonic sources, relying on a (possibly online) pitch detection, and able to perform very flexible pitch and formant modifications.

5.4 Conclusion

Modifying the duration or the pitch of a sound is not as easy as it might seem at first sight. Simply changing the sampling rate only partially does the job. In this Chapter, we have implemented a phase-locked vocoder and tested it for time scale and pitch scale modification. We have seen that processing the signal in the STFT domain provides better results, although not perfect yet. One of the reasons for the remaining artifacts lies in the interpretation of the input signal in terms of partials, which are neither easy to spot, nor even always clearly defined (in transients, for instance). For pitch shifting, we have also implemented an all-spectral approach, which is simpler than using a phase coder followed by a sampling rate converter, but still exhibits artifacts.

5.5 References

- Arfib D, Keiler F, Zölzer U (2002) Time-frequency processing. In: DAFX: Digital Audio Effects. U. Zölzer, Ed. Hoboken (NJ): John Wiley & Sons
- Bagdasarian R (1958) The Chipmunk song (Christmas don't be late). Jet Records, UK
- Dolson M (1986) The phase vocoder: A tutorial. *Computer Music Journal*, 10-4, pp 14-27

- Dutoit T (1997) Time-domain algorithms. In: An Introduction to Text-To-Speech Synthesis. Dutoit T. Dordrecht: Kluwer Academic Publishers
- Flanagan L, Golden RM (1966) Phase Vocoder. Bell System Technical Journal pp 1493–1509 [online] Available: <http://www.ee.columbia.edu/~dpwe/e6820/papers/FlanG66.pdf> [07/06/2007]
- Laroche J (1998) Time and pitch-scale modification of audio signals. In: Applications of Digital Signal Processing to Audio Signals. M. Kahrs and K. Brandenburg, Eds. Norwell, MA: Kluwer Academic Publishers
- Laroche J, Dolson M (1999) Improved Phase Vocoder Time-Scale Modification of Audio. IEEE Trans. Speech and Audio Processing, 3, pp 323–332
- Laroche J, Dolson M (1999b) New Phase Vocoder Technique for Pitch-Shifting, Harmonizing and Other Exotic Effects. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics. Mohonk, New Paltz, NY. [online] Available: <http://www.ee.columbia.edu/~dpwe/papers/LaroD99-pvoc.pdf> [07/06/2007]
- McAulay R, Quatieri T (1986) Speech analysis/Synthesis based on a sinusoidal representation. IEEE Transactions on Acoustics, Speech, and Signal Processing 34:4, pp 744–754
- Puckette MS (1995) Phase-locked Vocoder. Proc. IEEE Conf. on Applications of Signal Processing to Audio and Acoustics, Mohonk
- Röbel A (2003) A new approach to transient processing in the phase vocoder. Proc. 6th Int. Conference on Digital Audio Effects (DAFx-03), pp 344–349
- Sethares WA (2007) Rhythm and Transforms. London: Springer Verlag [online] Available: <http://eceserv0.ece.wisc.edu/~sethares/vocoders/Transforms.pdf> [20/3/08]
- Smith JO (2007) Example of Overlap-Add Convolution. In: Spectral Audio Signal Processing, Center for Computer Research in Music and Acoustics (CCRMA), Stanford University [online] Available: http://ccrma.stanford.edu/~jos/sasp/Example_Overlap_Add_Convolution.html [14/06/07]
- Smith JO (2007b) Dual of Constant Overlap-Add. In: Spectral Audio Signal Processing, Center for Computer Research in Music and Acoustics (CCRMA), Stanford University [online] Available: http://ccrma.stanford.edu/~jos/sasp/Example_Overlap_Add_Convolution.html [14/06/07]
- Verhelst V, van Compernelle D, Wambacq P (2000). A unified view on synchronized overlap-add methods for prosodic modifications of speech. In Proc. ICSLP2000, Vol. 2, 63–66