Pitch-shifting algorithm design and applications in music

THÉO ROYER
Abstract

Pitch-shifting lowers or increases the pitch of an audio recording. This technique has been used in recording studios since the 1960s, many Beatles tracks being produced using analog pitch-shifting effects. With the advent of the first digital pitch-shifting hardware in the 1970s, this technique became essential in music production. Nowadays, it is massively used in popular music for pitch correction or other creative purposes. With the improvement of mixing and mastering processes, the recent focus in the audio industry has been placed on the high quality of pitch-shifting tools. As a consequence, current state-of-the-art literature algorithms are often outperformed by the best commercial algorithms. Unfortunately, these commercial algorithms are “black boxes” which are very complicated to reverse engineer.

In this master thesis, state-of-the-art pitch-shifting techniques found in the literature are evaluated, attaching great importance to audio quality on musical signals. Time domain and frequency domain methods are studied and tested on a wide range of audio signals. Two offline implementations of the most promising algorithms are proposed with novel features. Pitch Synchronous Overlap and Add (PSOLA), a simple time domain algorithm, is used to create pitch-shifting, formant-shifting, pitch correction and chorus effects on voice and monophonic signals. Phase vocoder, a more complex frequency domain algorithm, is combined with high quality spectral envelope estimation and harmonic-percussive separation to design a polyvalent pitch-shifting and formant-shifting algorithm. Subjective evaluations indicate that the resulting quality is comparable to that of the commercial algorithms.
Sammanfattning


Acknowledgements

I would like to thank everyone who helped and supported me during this master thesis.

First and foremost, I am very grateful to Raphaël and Mickaël, my supervisors at Eiosis, for their continuous guidance and insight throughout the degree project, and also towards Mathieu who shared my office and was helpful many times. I wish to thank my KTH supervisor Saikat Chatterjee for the support and advices. And obviously, many thanks to everyone at the company for the positive working environment which I think greatly contributed to the quality of the work.
# Contents

1 Introduction ................................................. 1
   1.1 Context .............................................. 1
   1.2 Objective and outline ............................... 2

2 Technical background ...................................... 3
   2.1 Fourier analysis .................................... 3
      2.1.1 Discrete Fourier Transform .................. 3
      2.1.2 Windowing ...................................... 3
   2.2 Time-Frequency analysis ............................. 6
      2.2.1 Short-Time Fourier Transform ............... 6
      2.2.2 Constant overlap-add constraint ............ 6
   2.3 Introduction to pitch-shifting ..................... 8
      2.3.1 Fundamental frequency, harmonics, formants .. 8
      2.3.2 Pitch-shifting and formant-shifting ......... 9
      2.3.3 Relation between pitch-shifting and time-stretching 10
   2.4 Audio quality criteria .............................. 12
      2.4.1 Expected quality of pitch-shifting .......... 12
      2.4.2 Audio artifacts ................................ 12

3 State-of-the-art ............................................. 15
   3.1 Time-domain methods ................................. 15
      3.1.1 OverLap-Add .................................. 15
      3.1.2 Time-Domain Pitch-Synchronous OverLap-Add ... 18
   3.2 Frequency-domain methods ........................... 20
      3.2.1 Phase vocoder .................................. 20
      3.2.2 Phase-locked vocoder ......................... 29
      3.2.3 "Phase vocoder done right" .................. 29
      3.2.4 Transient preserving phase vocoders ........ 31
      3.2.5 Multi-resolution phase vocoders ............ 32
   3.3 Evaluation of pitch-shifting methods ............... 35
# 4 Applications

## 4.1 Voice correction, pitch and formant-shifting algorithm based on TD-PSOLA

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.1 Pitch detection</td>
<td>36</td>
</tr>
<tr>
<td>4.1.2 Pitch post-processing</td>
<td>36</td>
</tr>
<tr>
<td>4.1.3 Formant-shifting</td>
<td>40</td>
</tr>
<tr>
<td>4.1.4 Voice correction</td>
<td>41</td>
</tr>
<tr>
<td>4.1.5 Chorus effect</td>
<td>44</td>
</tr>
<tr>
<td>4.1.6 Implementation</td>
<td>46</td>
</tr>
<tr>
<td>4.1.7 Results</td>
<td>46</td>
</tr>
</tbody>
</table>

## 4.2 Polyvalent transient preserving pitch-shifting and formant-shifting algorithm based on the phase vocoder

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.1 Pitch-shifting method</td>
<td>47</td>
</tr>
<tr>
<td>4.2.2 Formant-shifting with spectral envelope estimation</td>
<td>47</td>
</tr>
<tr>
<td>4.2.3 Transient preservation</td>
<td>50</td>
</tr>
<tr>
<td>4.2.4 Implementation</td>
<td>55</td>
</tr>
<tr>
<td>4.2.5 Results</td>
<td>57</td>
</tr>
</tbody>
</table>

# 5 Conclusions and future work

## 5.1 Conclusions

## 5.2 Future work

# Bibliography

# A Amplitude flatness

# B Details on the "phase vocoder done right" algorithm

## B.1 Algorithm

## B.2 A simple example

# C Details on pitch correction smoothing

# D Graphical User Interface of pitch-shifting algorithms

# E Harmonic-Percussive separation algorithm

# F Pre-echo reduction processing
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Frequency response of a sin wave with different window sizes</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>Comparison between DFT magnitudes of a sum of 2 sinusoids analyzed through rectangular and Hamming windows</td>
<td>5</td>
</tr>
<tr>
<td>2.3</td>
<td>Example of frames extracted from an audio signal with a Hanning window, analysis size = 1024 samples and hop size = 512 samples</td>
<td>7</td>
</tr>
<tr>
<td>2.4</td>
<td>Spectrogram of an extract from Bohemian Rhapsody</td>
<td>7</td>
</tr>
<tr>
<td>2.5</td>
<td>Illustration of amplitude flatness on Hanning windows with different overlap ratios</td>
<td>8</td>
</tr>
<tr>
<td>2.6</td>
<td>Illustration of fundamental frequency f0, spectrum (red), spectrum envelope (black) and formants F1, F2, F3</td>
<td>9</td>
</tr>
<tr>
<td>2.7</td>
<td>Theoretical pitch-shifting in the frequency domain, from [8]</td>
<td>11</td>
</tr>
<tr>
<td>2.8</td>
<td>Theoretical pitch-shifting with formants preservation in the frequency domain, from [9]</td>
<td>11</td>
</tr>
<tr>
<td>2.9</td>
<td>Pitch-shifting as a combination of time-stretching and resampling</td>
<td>12</td>
</tr>
<tr>
<td>2.10</td>
<td>Effect of transient duplication on a drums signal</td>
<td>13</td>
</tr>
<tr>
<td>2.11</td>
<td>Effect of transient smearing on a drums signal</td>
<td>14</td>
</tr>
<tr>
<td>2.12</td>
<td>Effect of clipping on a drums signal</td>
<td>14</td>
</tr>
<tr>
<td>3.1</td>
<td>Up-shifting example with OLA method</td>
<td>16</td>
</tr>
<tr>
<td>3.2</td>
<td>Waveform of up-shifted drums clap with OLA method</td>
<td>17</td>
</tr>
<tr>
<td>3.3</td>
<td>Block diagram of TD-PSOLA principle</td>
<td>18</td>
</tr>
<tr>
<td>3.4</td>
<td>Example of pitch values obtained over time on a voice signal</td>
<td>19</td>
</tr>
<tr>
<td>3.5</td>
<td>Waveform example of a signal up-shifted by TD-PSOLA, from [11]</td>
<td>20</td>
</tr>
<tr>
<td>3.6</td>
<td>Phase vocoder block diagram, from [9]</td>
<td>22</td>
</tr>
<tr>
<td>3.7</td>
<td>Filter-bank representation of the STFT, from [9]</td>
<td>23</td>
</tr>
<tr>
<td>3.8</td>
<td>Wrapped and unwrapped phase of a sinusoidal signal</td>
<td>24</td>
</tr>
<tr>
<td>3.9</td>
<td>Synthesis phase computation at a given frequency channel preserving horizontal coherence, ha = analysis hop size, hs = synthesis hop size</td>
<td>26</td>
</tr>
<tr>
<td>3.10</td>
<td>Example of pitch-shifting with the phase vocoder on a sine wave</td>
<td>27</td>
</tr>
</tbody>
</table>
3.11 Conceptual difference between phase propagation in the standard
phase vocoder, the phase-locked vocoder [20] and the phase vocoder
done right from [24]. .............................................. 30
3.12 Phase propagation paths, from [24], horizontal axis is time, vertical
axis is time, box darkness represents bin magnitude. ................. 32
3.13 Comparison between STFT and CWT spectrogram on a drums track . 33
3.14 CQT representation used for pitch-shifting, from [29] ............... 34
4.1 Difference between uncorrected and corrected DFT pitch estimate on
a voice signal, from [9]. ........................................... 37
4.2 FFT-based pitch estimation error relative to frequency. .............. 38
4.3 Time signal and its NMDF ....................................... 39
4.4 Comparison between standard and tapered NMDF, from [34] ....... 40
4.5 Pitch processing steps on a voice signal ................................ 42
4.6 Simple formant-shifting in TD-PSOLA. ............................ 43
4.7 Input pitch and corrected pitch of an extract from Bohemian Rhapsody, without any smoothing. .......................... 44
4.8 Comparison between unprocessed and smoothed pitch-shifting factor
over time .......................................................... 45
4.9 Input pitch and corrected pitch of an extract from Bohemian Rhapsody, with smoothing ............................................ 45
4.10 Waveforms of a transient pitch-shifted with standard phase vocoder
and "phase vocoder done right" ................................. 48
4.11 Oversampling effect on high frequency noise in the "phase vocoder
done right" ......................................................... 49
4.12 Pre-echo when pitch-shifting a transient ......................... 50
4.13 Computation of correction envelope based on input frame envelope
estimation ............................................................ 52
4.14 Peak-based envelope estimation method compared to cepstrum envelope
estimation ......................................................... 53
4.15 Peak blank-filling in concave part of the spectrum .................. 54
4.16 First iteration of the true spectrum estimation method ............ 55
4.17 True envelope estimation method .................................. 56
4.18 Waveform of extracted percussive component overlapped to the complete
input signal .......................................................... 57
4.19 Waveforms of down-shifted drum clap with harmonic-percussive separation
algorithm and commercial algorithm elastiquePro ............... 58
A.1 Amplitude flatness (pink) in function of overlap ratio for Hanning and
Kaiser windows ..................................................... 64
D.1 GUI of pitch-shifting/Autotune voice tool ................................ 74
D.2 GUI of voice chorus tool .......................... 74
D.3 GUI of phase vocoder pitch-shifting tool .............. 75
D.4 GUI of transient preserving pitch-shifting tool ......... 75
F.1 Block diagram of pre-echo reduction processing ......... 83
Chapter 1

Introduction

1.1 Context

Pitch-shifting is the operation which changes the pitch of a signal without altering its length. First pitch-shifter analog hardware was designed in the 1950s. It worked by recording the signal on a tape at a certain speed but using a different head tape speed when reading than when recording. The reading head speed was controlled by a keyboard, the audulator [1], which defined by how much the pitch was increased or decreased. Similarly, time-stretching, the operation which changes the length of a recording without altering its pitch, was done by changing the tape speed instead of the head speed. These devices were more commonly used for time-stretching radio commercials so that they can fit the required length rather than for pitch-shifting music signals. Some of the rare examples music tracks mentioned are from the Beach Boys [2].

In the 1950s and 1960s, pitch-shifting was typically done by changing the reading speed. By doing so, the tempo was also changed. It was not a real pitch-shifting technique by itself but some interesting transformations using this technique were achieved. This operation is also called “Varispeed”. By recording a signal at a lower tempo and then increasing the tempo so that it matches with the target tempo, the pitch is increased. Alvin and the Chipmunks were recorded using this method [3]. Varispeed was used in many of the Beatles songs [4]. First subtle application was to change the timbre of the voice. For some songs, vocals were recorded at a slightly lower pitch and slower than the rest of the song. When sped up to the original speed, the pitch increases and corresponds to the original pitch and tempo of the song. However, the timbre of the voice is changed. A similar but less subtle effect was obtained to change a piano timbre into a harpsichord timbre. Because of the way these transformations are obtained, it was only possible to change the pitch or the timbre as an offline process.
Eventide H910 Harmonizer, the first real-time digital pitch-shifting hardware, was released in 1975 \[5\]. It quickly became a standard tool for creating unique sound effects at the time. Examples of famous artists using the harmonizer in the late 70s to 80s are David Bowie, Van Halen or U2 \[6\]. Since then, many similar products and upgrades of the harmonizer have been released. Now, virtual plugins replaced hardware for pitch-shifting and the need for high quality has never been so high. Pitch-correction is widely used in popular music and creative applications of pitch-shifting are more numerous and better known than at the release of the Harmonizer. Developing a high quality pitch-shifter is very challenging as this transformation can bring many audio artifacts (see section 2.4.2). This is why the reference commercial algorithms such as Autotune, Melodyne and Elastique have emerged from highly specialized audio companies. On the other hand, open literature resources are nowadays outperformed by these commercial algorithms.

1.2 Objective and outline

This degree project was carried out at a company named Slate Digital which designs audio plugins for music production used in digital audio workstations (DAW). DAWs are softwares used in audio production contexts such as music, television or radio. The objective of this project is to study the existing pitch-shifting techniques from the literature, implement and improve the most promising ones for musical applications. As explained later in the report, the focus is placed on a high output audio quality. The implementations of the algorithms are offline but the algorithms should be reasonably efficient to be implemented in a real-time framework. The report is organized as follows. Chapter 2 presents some signal processing definitions and notions related to pitch-shifting. Chapter 3 provides an extensive state-of-the-art and some results of the most promising time domain and frequency domain pitch-shifting methods. Chapter 4 details the implementations of 2 pitch-shifting applications. Chapter 5 summarizes the results and provides insight on future work.
Chapter 2

Technical background

2.1 Fourier analysis

2.1.1 Discrete Fourier Transform

The Discrete Fourier Transform (DFT) of a signal is its decomposition into a sum of complex sinusoids, the spectrum. The DFT of a discrete and finite signal $x[n], 0 \leq n \leq N - 1$ is mathematically defined as:

$$X[k] = \sum_{n=0}^{N-1} x[n].e^{-i2\pi nk/N}$$  \hspace{1cm} (2.1)

The input signal and its DFT have the same length $N$. Each coefficient $X[k]$ of the DFT, also referred to as bin from frequency channel $k$, relates to a complex sinusoid whose normalized frequency is $k/N$. The magnitude of a bin defines the magnitude of the corresponding sinusoidal component in the input signal. The phase of a bin defines the time offset of the sinusoidal component. The DFT is perfectly invertible, which means the original time signal can be reconstructed identically from the frequency coefficients with the inverse Discrete Fourier Transform (iDFT), if the coefficients remain unchanged in the frequency domain.

2.1.2 Windowing

To analyze a finite time interval of a signal, a windowing function is applied on it. It consists in multiplying the signal by a window which is only non zero on a studied interval. Windowing has an impact on spectral estimation. This phenomenon is called the uncertainty principle and can be observed in many different fields. In signal processing, we are limited in localizing the signal both in time and frequency domain. If we use a very wide window in time domain, we are able to localize it very well in frequency but not in time. Similarly, if we use a very narrow window in time domain, we can localize it very well in time but not in frequency. It is illustrated
in figure 2.1, where a sine wave is clearly better identified in the frequency domain when using a 1024 samples window than a 256 samples window in time domain.

The shape of the window also plays a significant role. The rectangular window (no window) is the window which minimize the width of the main lobe in the frequency domain. However, the side lobes have a high amplitude. Most windows used in signal processing have a bell shape and are equal to 0 at their borders. Figure 2.2 shows the effect of windowing when analyzing a sum of 2 sinusoids in the frequency domain. The Hamming window is clearly better at analyzing the 2 distinct sinusoidal components as the maximum magnitude of the peaks are way above the rest of the spectrum, as opposed to the spectrum of the rectangular windowed signal.
Figure 2.2: Comparison between DFT magnitudes of a sum of 2 sinusoids analyzed through rectangular and Hamming windows
2.2 Time-Frequency analysis

2.2.1 Short-Time Fourier Transform

The DFT provides a time-fixed frequency representation of a signal. Music frequency content is highly varying so the DFT has to be computed at different times in order to give relevant information. Considering the DFT size \( N \), a window function \( w[n] \) which is only non zero where \(-N/2 \leq n \leq N/2\) and a \( N_s \)-sample long signal \( x[n] \), the Short-Time Fourier Transform (STFT) adds the time dimension and is defined by:

\[
X[k, t] = \sum_{n=0}^{N_s-1} x[n] w[n-t] e^{-i2\pi nk/N}
\]  

(2.2)

It can be interpreted as several DFTs computed on a signal multiplied by a window sliding over time. I will refer to these windowed versions of the signal as frames. Each time we slide the window and multiply it to the input signal, we get a new frame and compute its DFT. The way we extract these frames depends on 3 parameters:

- the window type: rectangular, Hanning, Blackman, Kaiser ....
- the analysis window size: it defines the size of the resulting frames in samples.
- the hop size: it is the step between 2 consecutive frames in samples.

On figure 2.3, 3 consecutive frames are plotted below their original input signal with hop size being half the analysis size. By computing the DFT of each of these frames and placing the result into a 2D-array, we obtain the STFT. The STFT can be displayed as an image, the spectrogram (see figure 2.4) by taking the magnitude of each frequency bin and assigning a color based on the magnitude value.

2.2.2 Constant overlap-add constraint

Constraints exist on the type of window, the analysis window size and the hop size chosen in order to obtain perfect reconstruction when using the iSTFT. It is called in the literature the constant overlap-add constraint or amplitude flatness [7]. This constraint states that, when summing all the windows, we must obtain a flat amplitude. Depending on the type of windows, relations between analysis window size and hop size have to be respected.
Figure 2.3: Example of frames extracted from an audio signal with a Hanning window, analysis size = 1024 samples and hop size = 512 samples

Figure 2.4: Spectrogram of an extract from Bohemian Rhapsody
We define the overlap ratio as $1 - \frac{h}{a}$, where $a$ is the analysis window size and $h$ is the hop size. For instance, the overlap ratio is $1/2$ if $h = 512$ and $a = 1024$, $3/4$ if $h = 512$ and $a = 2048$, etc. Some examples of overlap ratios respecting amplitude flatness for Hanning windows are $1/2$, $2/3$, $3/4$. Using different values results in modulated amplitude as shown in figure 2.5. For some specific windows such as the Kaiser windows, amplitude flatness cannot be mathematically achieved but high overlap ratios are chosen to obtain an almost flat amplitude such as this is imperceptible. More details on windows and amplitude flatness can be found in appendix A.

2.3 Introduction to pitch-shifting

2.3.1 Fundamental frequency, harmonics, formants

The fundamental frequency of a sound, also referred to as pitch, is the lowest frequency component of its waveform, noted $f_0$. Harmonics are components whose frequencies are multiples of the fundamental frequency, noted $f_k$. While the fundamental frequency only defines if a sound has a high or low pitch, harmonics amplitudes define the timbre: is it voice, guitar, drums? Maxima in the spectrum
envelope are referred to as formants for voice. Formants can also be defined for acoustic instruments which work in a similar way than voice.

These concepts are more easily explained with figure 2.6. In the frequency domain, we can observe the spectrum. It is the magnitude of the DFT (in red). The spectrum is very fast varying in frequencies and it shows magnitude peaks at each harmonic frequency. We can also define the spectral envelope (in black), a smooth curve following the harmonic peaks. The maxima of the spectral envelope define the formants frequencies. Formants characterize the envelope and the envelope characterizes the timbre of a sound. For one person's voice, these formants remain the same for different fundamental frequencies. This is why we can distinguish 2 voices at the same fundamental frequency and also recognize a single voice singing at 2 different fundamental frequencies. Similar observations can be made on acoustic instruments in general.

2.3.2 Pitch-shifting and formant-shifting

Pitch-shifting is changing the tone of a sound to a higher tone (up-shifting) or a lower tone (down-shifting). In music, the semitone is the most commonly used smallest interval of pitch. Our perception of pitch is based on a logarithmic scale so a semitone does not correspond to a fixed frequency difference. Considering a note whose fundamental frequency is $f_0$, the frequency of the note which is $k$ semitones higher (positive $k$) or lower (negative $k$) is $f_0 \cdot 2^{k/12}$. Taking the reference note A440, the A note whose fundamental frequency is $440 \text{Hz}$, the next note which is one semitone higher has a fundamental frequency of $440 \cdot 2^{1/12} = 466 \text{Hz}$. This corresponds to a $26 \text{Hz}$ frequency shift. The next A note, A880, has a fundamental frequency
of 880 Hz. The note which is one semitone higher has a fundamental frequency of \(880 \times 2^{1/12} = 932\) Hz. This corresponds to a 52 Hz frequency shift. This difference illustrates that a semitone is not a fixed frequency shift on a linear scale.

A similar frequency transformation is frequency shifting. It consists in shifting the spectrum by a fixed amount of frequency. If the original spectrum is noted \(S(f)\), the new frequency-shifted spectrum is \(S_{freq-shift} = S(f - f_0)\). This is an easy operation which is equivalent to amplitude modulation. However, this cannot be used for pitch-shifting because it would break the relations between notes. Shifting the spectrum by 100 Hz would constitute a several octave shift for low frequencies but only a few semitones shift for high frequencies.

To preserve pitch relationships, the spectrum needs to be scaled or dilated. The new pitch-shifted spectrum is \(S_{pitch-shift} = S(f/\beta)\), where \(\beta = 2^{k/12}\) is the pitch-shifting factor of \(k\) semitones. An ideal pitch-shifting operation is illustrated in figure 2.7. What can be seen is that, as expected, the entire spectrum is scaled, both the fast varying spectrum and its envelope. Because the pitch is higher on the transposed sound, the space between the harmonics is also larger. However, the envelope is also scaled so the formants positions are different. In this case, we also shifted the formants. It can be a problem if we apply pitch-shifting to voice because this would change the timbre. Non-formant preserving pitch-shifting techniques give a chipmunk effect on voice when up-shifting and a dark villain voice when down-shifting. Some pitch-shifting techniques preserve formants, changing only the fundamental frequency and harmonics positions while preserving the envelope. For pitch-shifting algorithms that do not preserve formants, some other techniques can be used to only shift the formants in order to correct or change the timbre. The effect of a formant preserving pitch-shifting operation is illustrated on figure 2.8.

2.3.3 Relation between pitch-shifting and time-stretching

Pitch-shifting is the operation that consist in changing the pitch without changing the duration of a sound. Time-stretching is the opposite, it changes the duration of a sound without changing its pitch. These 2 operations are similar and a lot of pitch-shifting algorithms are based on time-stretching algorithms combined with resampling. If we want to transpose the pitch by a factor \(\beta\), we can first time-stretch the signal so that the time-stretched signal length is \(N \times \beta\) where \(N\) is the original length of the signal. Then we resample the signal whose length is \(N \times \beta\) to have a signal whose length is \(N\). On the left side of figure 2.9, we upshift the signal by an octave so the transposition factor is 2. The signal is time-stretched so that we double its length. Then we resample the signal so that it is twice as short. Down-shifting is computed similarly.
Figure 2.7: Theoretical pitch-shifting in the frequency domain, from [8].

Figure 2.8: Theoretical pitch-shifting with formants preservation in the frequency domain, from [8].
2.4 Audio quality criteria

2.4.1 Expected quality of pitch-shifting

With the idea of designing a pitch-shifting algorithm which could be used in professional music environments, the expected audio quality of the operation is high. In this context, some aspects that would be viewed as secondary in pure academic research are much more important here. Focus is set on having the best perceptual quality rather than optimizing a mathematical criterion such as the signal-to-noise ratio, the main reason being that it is still hard to mathematically define what sounds good or not. It is thus not easy to describe why a solution might be better than another solution through a report, because the motivation would be: "It sounds better". The purpose of this section is to describe at best the audio artifacts that I encountered when designing pitch-shifting algorithms, so that the motivation behind the choices I had to make during this project are more clear to someone who can only read the report.

2.4.2 Audio artifacts

This section presents a non exhaustive list of audio artifacts encountered in pitch-shifting algorithms.

- Detuning: When audio components are not "in tune". Instead of changing the pitch of all the frequency components by the same number of semitones, some are a little bit more shifted than others. The effects are ranging from an almost imperceptible strange feeling to a completely unlistenable audio track.
• Chorus : When several signal sources are perceived instead of one. On voice, it is hearing voice duplicates instead of only one person.

• Transient duplication : When a transient (short aperiodic sound) such as a drums kick is repeated twice in a very short period, see figure 2.10.

• Transient smearing : When a transient attack, the first abrupt increase, is softened, see figure 2.11.

• Pre-echo : Reduced version of transient smearing, a very short chirp sound heard a few ms before transients.

• Clipping : When the signal goes above the maximum coded value, it is clipped at the maximum value and this saturation effect generates high frequency components, see figure 2.12.

• Resonance : When signal components oscillate longer than they should be on the original signal.

• Phasiness or loss of presence : When the sound feels weaker, distant and much less dynamic overall, typical artifact from the phase vocoder.

• Modulation : When a vibration at a fixed frequency can be heard continuously, by modulating the input signal.
Figure 2.11: Effect of transient smearing on a drums signal

Figure 2.12: Effect of clipping on a drums signal
Chapter 3

State-of-the-art

There are 2 main types of pitch-shifting methods. Time domain algorithms rely on simple transformations of the signal which make them easy to implement at a low computational cost. Frequency domain algorithms are based on frequency analysis and transformations in the frequency domain. They are more computationally intensive and also less intuitive, however they allow more complex transformations of the signals. Both types of methods have their advantages and disadvantages which will be discussed in the next sections.

3.1 Time-domain methods

3.1.1 OverLap-Add

Method

OverLap-Add (OLA) is the most basic way to do time domain pitch-shifting. Similarly to the STFT, overlapping frames of signal are extracted. An important aspect of OLA algorithms is that they naturally preserve the formants unlike frequency domain methods. Two variants of OLA algorithm are used when up-shifting or down-shifting.

When up-shifting, the following pitch-shifting method is used. The synthesis signal is reconstructed by overlapping and adding frames with a smaller hop size. As we reduce the space between the frames, the synthesis signal becomes shorter. To keep the same signal length, some analysis frames are duplicated in the synthesis signal. An example is illustrated on figure 3.1.

Down-shifting would result in an increase in the space between frames and decrease in amplitude flatness. This is because amplitude flatness is more affected when reducing the overlap between frames than when increasing. This is explained
Figure 3.1: Up-shifting example with OLA method
A slightly different method can be used to solve this problem when down-shifting by time-stretching and resampling. A frame-by-frame implementation of time-stretching and resampling for OLA uses a synthesis hop size equal to the analysis hop size and resample each frame so that its length is divided by the transposition factor $\beta$. $\beta \leq 1$ when down-shifting, so each frame is dilated, resulting in what can be perceived as a decrease in the pitch.

**Performances and limits of simple OLA**

This algorithm works well under the assumption that the window size corresponds to the fundamental period of the sound we want to pitch-shift. It cannot be used for sounds with a rich and wide spectrum because it cannot work well for low and high frequencies at the same time. This algorithm is easy to implement but not flexible because everything is fixed: same fixed window size for the analysis and synthesis, fixed hop size for the analysis and fixed hop size for the synthesis. By testing it, it was obvious that it did not work well for any signal having long and periodic component such as voice, guitar or bass. However, it provides decent results on drums (see 3.2), as long as we pitch-shift the signal by only a few semitones. On any monophonic content, modulation and chorus effects can be heard with small shifting and it fails completely once the shift is greater than 6 semitones.
3.1.2 Time-Domain Pitch-Synchronous OverLap-Add

Introduction to PSOLA

The main issue with simple OLA is that the size of the windows is fixed and as a result, it cannot be well adapted to the pitch of the input signal. Time-Domain Pitch-Synchronous OverLap-Add (TD-PSOLA) is a more refined version of the OLA method specialized for monophonic signal, presented in [10] and [11], which solves the previous problem. TD-PSOLA method relies on a source-filter model as shown on figure 3.3. It supposes the signal is generated by a frequency varying impulse train filtered by a varying model. By determining the pitch of the signal over time, the generative impulse train can be extracted as well as each filter model. The filter model is the frame extracted, it determines the timbre of the sound. The impulse train determines its pitch. If we generate the synthesis signal by increasing the impulse train frequency while keeping the same frames, then we can pitch-shift the input without affecting the formants.

Method

First step of the algorithm is to determine pitch values in small signal frames, typically 1024 samples long with $f_s = 44.1$kHz. Some pitch detection methods are explained in section 4.1. Pitch values are obtained over time as presented in figure 3.4. Then pitchmarks are positioned at each period of the signal. Here, pitch mark-
ing is simply done by setting the space in sample between 2 pitchmarks as the period of the pitch detected. This does not guarantee the positioning of the pitchmarks at local energy maxima but it was not found to be particularly important to center the pitchmarks on the energy peaks, even though it was suggested in the literature. It is said in \cite{12} and \cite{9} that a more robust pitch marking can improve transient preservation. After pitchmarking, frames are extracted such as their first sample is located at a pitchmark k-1 and their last sample is located at the pitchmark k+1. This way, each frame includes 2 periods of the signal.

The actual pitch-shifting operation is slightly more complicated, as the analysis and synthesis sizes are not fixed but depends on the detected pitch. Synthesis pitchmarks are built from analysis pitchmarks. The space between pitchmarks increases when down-shifting, and decreases when up-shifting. \( P^a_k \) being the position of the \( k^{th} \) analysis pitchmark and \( P^s_k \) the position of the \( k^{th} \) synthesis pitchmark, the position of the \( k + 1^{th} \) synthesis pitchmark is computed as follows:

\[
P^s_{k+1} = P^s_k + \frac{(P^a_{k+1} - P^a_k)}{\beta}
\]  

Mapping between analysis frame and synthesis frame is the same as in the OLA algorithm. For each synthesis pitchmark, we find the closest analysis pitchmark and take the corresponding analysis frame to be used as the synthesis frame at the synthesis pitchmark. The main difference with the simple OLA algorithm is that analysis and synthesis frame positioning cannot be predetermined as they depend
Results

TD-PSOLA is very effective on monophonic tracks. Tests on violin showed that as soon as the musician played 2 notes simultaneously, pitch detection tended to jump between the 2 notes. While one note was correctly shifted, the other was not. Moreover, pitch-shifting quality is directly correlated to pitch detection quality. The most obvious application for TD-PSOLA is voice pitch-shifting as the method is designed to preserve formants, thus not changing the timbre of the voice when pitch-shifting. A TD-PSOLA implementation inspired by [13] and [14] are further discussed in 4.1 with more details about how pitch detection and pitch marking are designed.

3.2 Frequency-domain methods

3.2.1 Phase vocoder

Introduction to the phase vocoder

The phase vocoder is an analysis-synthesis process that can be used to make time-frequency modification of an audio signal. The input signal is analyzed to extract magnitude and phase for different frequency components. Phase information is then transformed, and the output signal is reconstructed using the magnitude and
the new phase.

Historically, this process was done in the 1960s by using filter banks \[15\], where each filter isolates a narrow frequency band of the signal. For each filter, the magnitude and phase of the signal are extracted. New phase values are computed and a sinusoid is generated with an oscillator using the magnitude and the new phase. The output signal is reconstructed as a sum of sinusoids. There is a more common way to do this by using the Short Term Fourier Transform (STFT) \[16\]. STFT is computed by sliding a DFT on different frames extracted from an overlapping frame scheme. For each frame we compute the DFT and take the magnitude and phase, compute the new phase and reconstruct the complex signal using the magnitude and new phase. Then, the frame is reconstructed at the synthesis using IDFT. The output is reconstructed by summing the overlapping reconstructed frames. This process is shown on figure 3.6 from \[9\].

Both approaches are very similar because we can see the STFT as a filter-bank operation. Each new time index of the STFT corresponds to a new time \(t\) when the signal is analyzed. Each frequency bin \(k\) of the DFT corresponds to the output of a band pass filter giving the magnitude and the phase of a sinusoid of frequency \(f_k\). Also, frame reconstruction through IDFT is similar to adding outputs of oscillators with different frequencies, magnitudes and phases.

On figure 3.7 is shown a filter-bank representation of the STFT, where each line corresponds to the output of the frequency channel \(k\) over time and each column corresponds to the output of all the frequency channels at a given instant. Each line can be interpreted as a band-pass filter. The number of frequency channels \(N\) of the DFT (or band-pass filters) is the size of the analysis window of the STFT.

**Issues related to pitch transformations in the frequency domain**

If, for example, we take a window size of 1024 samples, it is as if we had 1024 band-pass filters to analyze our signal. 1024 can be seen as a lot of filters, but for an audio signal with a sampling rate of 44.1kHz, having 1024 filters equally spaced means that the frequency resolution is 43Hz.

The 3rd and 4th frequency channels of the STFT are centered around 86Hz and 129Hz. What happens if we analyze a pure sinusoidal signal whose frequency is 100Hz? The bin will be spread over the adjacent bins at 86Hz and 129Hz. In the end, what we see in the frequency domain is not a sinusoidal signal whose frequency is 100Hz, but a sum of 2 sinusoids of frequencies 86Hz and 129Hz.

As long as we don’t change anything to the frequency representation, it is not
Figure 3.6: Phase vocoder block diagram, from [9]
a problem since we are able to reconstruct the original sinusoidal signal perfectly. However, if we want to do pitch modifications of this 100Hz signal, it can be complicated because, instead, what we see in the frequency domain is a sum of sinusoids with different frequencies. This uncertainty in the frequency content is what makes transformations in the frequency domain tricky.

**Phase unwrapping and instantaneous frequency**

As its name suggests, the phase vocoder does time-frequency modification by recomputing the phase of the frequency bins. A little more background is required on phase unwrapping and instantaneous frequency to understand how the phase vocoder works.

We mostly refer to the phase of a sine wave as its principal value in the interval $[-\pi, \pi]$, the wrapped phase. However, we can also consider the unwrapped phase of a signal which is continuous and unbounded. If $x(t) = \cos(2\pi ft + \phi)$, the unwrapped phase of $x(t)$ is $2\pi ft + \phi$. Difference between wrapped and unwrapped phase is shown on figure 3.8. In the phase vocoder, we want to know the unwrapped phase difference at one frequency channel between 2 consecutive frames. Imagine we have a spinning wheel, and we know at which speed it spins or at least have a good estimate. We first note the angle of the wheel before it spins, then let it spin for a known period of time and check what is the angle of the wheel when it stops.
The task is to know by how much the wheel spun in total.

Let the starting angle be \( \phi(k, n) \), where \( k \) defines the wheel spinning frequency \( f_k \) in Hz and \( n \) the time index. We define the time interval between the 2 measurements \( h \) in s. We know the approximate speed of the wheel and the interval between the 2 measurements, so the best guess for the unwrapped angle is the target angle:

\[
\phi_t(k, n + 1) = \phi(k, n) + h.2\pi f_k
\] (3.2)

By comparing to the target angle to the real measurement \( \phi(k, n + 1) \), we can know the real unwrapped angle:

\[
\phi_u(k, n + 1) = \phi_t(k, n + 1) + \text{princarg}(\phi(k, n + 1) - \phi_t(k, n + 1))
\] (3.3)

where \( \text{princarg}(x) \) is the bounded value of \( x \) between \(-\pi\) and \( \pi \), \( \text{princarg}(x) = (x + \pi) \mod (2\pi) - \pi \).

We can finally compute the true unwrapped angle difference between the measurements \( \phi(k, n) \) and \( \phi(k, n + 1) \):

\[
\Delta \phi(k, n+1) = \phi_u(k, n+1) - \phi(k, n) = h.2\pi f_k + \text{princarg}[\phi(k, n+1) - \phi(k, n) - h.2\pi f_k]
\] (3.4)

Now, let’s consider that each wheel is a frequency channel of the STFT and that each measurement is a new time index of the STFT, then we computed the unwrapped phase difference \( \Delta \phi(k, n + 1) \) between 2 consecutive bins. The instantaneous frequency is defined for a frequency channel \( k \) as the phase derivative with
respect to time. $\Delta \phi(k, n+1)/(2\pi h)$ is the instantaneous frequency in Hz for the $k^{th}$ bin. The unwrapped phase difference is proportional to the instantaneous frequency. This operation gives us information on how fast phase evolves between 2 consecutive frames.

**Pitch-shifting by time-stretching and resampling with the phase vocoder**

Pitch-shifting cannot be done directly with the phase vocoder, so instead we use time-stretching and resampling (see section 2.3.3). Time-stretching in the phase vocoder works by using different hop sizes at analysis and synthesis. By increasing or decreasing the step between frames, we can increase or decrease the length of the output signal. However, to obtain the expected results, this is not enough.

Time-stretching in the phase vocoder is based on instantaneous frequency preservation while changing the step between consecutive frames. Assuming we want to time-stretch the signal by a factor $\alpha$, the synthesis hop size is the analysis hop size multiplied by $\alpha$. If $\alpha > 1$, the synthesis hop size is larger than the analysis hop size and we obtain a longer signal. Although, the instantaneous frequency is lower because, for a given frequency channel $k$, the phase difference between 2 consecutive bins in time is the same while the time difference between 2 consecutive bins in time is larger. In order to keep the same instantaneous frequency, we must multiply it by $\alpha$ at the synthesis.

We consider $\phi(k, n)$ (respectively $\phi_s(k, n)$), the phase of the bin at the $k^{th}$ frequency channel of the $n^{th}$ time frame at the analysis (respectively synthesis). $\Delta \phi(k, n+1)$ is computed from $\phi(k, n)$ and $\phi(k, n+1)$ as defined in equation 3.4. The synthesis phase is initialized for the first frame such as $\phi_s(k, 0) = \phi(k, 0)$. The next synthesis phases are computed as follows:

$$\phi_s(k, n+1) = \phi_s(k, n) + \Delta \phi(k, n+1).\alpha$$

Doing so, we maintain the same instantaneous frequency in the output while increasing the length of the signal. This preserves the horizontal phase coherence, which states that for a given frequency channel $k$, there is no phase discontinuity along the time axis. This process is illustrated by figure 3.9. The phase difference between the 1st and 2nd synthesis frames is higher than the phase difference between the 1st and 2nd analysis frames but because the time interval between 2 frames is larger, the instantaneous frequency represented by the slope remains the same. This is horizontal phase propagation.

Pitch-shifting can be obtained by time-stretching and resampling. Little changes can be made to this algorithm to transform the time-stretching algorithm into a pitch-shifting algorithm. A naive method would be to time-stretch the entire signal and then resample it to obtain the pitch-shifted signal. Such method would be impossible to use in a real-time framework. A frame-by-frame implementation of
the phase vocoder is explained in [9]. It follows the same steps than the time-stretching algorithm with some differences. Each frame is resampled so that its length is divided by $\alpha$. Because of the resampling operation, the synthesis hop size is also divided by $\alpha$. In the end, the analysis and synthesis hop size are the same.

**Example on a sine wave**

The best way to understand how pitch-shifting works with the phase vocoder is to show an example on the most simple signal we can pitch-shift: a sine wave. The example is illustrated by figure 3.10. In this example, we increase the pitch of the input signal by an octave so the pitch transposition factor is $\beta = 2$. We consider that the sine wave is seen in the frequency domain as a pure sine wave, which means that its spectrum is a Dirac at the $k^{th}$ frequency channel. Note that this is not possible except when using infinitely long windows and that this approximation is made to simplify the example. The analysis size is the period of the sine wave and the hop size is half the analysis size.

The first 2 frames of the signal are extracted through a rectangular window. At the next step, the phase of the $k^{th}$ bin needs to be changed. The 1st frame remains unchanged because of the initialization process, so $\phi_s(k, 0) = \phi(k, 0) = 0$. Then the unwrapped phase difference $\Delta \phi(k, n + 1)$ is computed. There is a half-period offset between the 2 analysis frames so $\Delta \phi(k, n + 1) = \pi$. The synthesis phase of the 2nd
Figure 3.10: Example of pitch-shifting with the phase vocoder on a sine wave
frame can now be computed as follows: \( \phi_s(k, 1) = \phi_s(k, 0) + \Delta \phi(k, 1), \beta = 0 + 2\pi = 2\pi \). After that, each frame needs to be resampled so that its size is twice as short. Finally, the output is fully constructed by overlap-add. We obtain an output signal whose pitch is doubled. In this example, the output signal appears as shorter but it is because of the initialization process. If the input signal was 10 times as long, the size difference between output and input would still be the same.

**Implementation and limitations of the simple phase vocoder**

The phase vocoder pitch-shifting algorithm was implemented in Python as a frame-by-frame but offline process. Choice of parameters was made according to personal testing and implementation tutorials [17] and [18]. A framework for a real-time implementation is also discussed in [19] but is not the main concern here. The following optimal parameters are defined for a sampling rate of 44.1kHz:

- analysis window size : 2048 samples
- hop size : 256 samples
- window choice : Kaiser with parameter \( \alpha = 10 \)

Various tests were conducted on a wide range of audio signals: voice, acoustic instruments, electronic music, drums, sinusoidal tones, etc. The pitch-shifting range of the phase vocoder is between -1 octave (-12 semitones) and +1 octave (+12 semitones). First realization is that the phase vocoder can pitch-shift any signal, from simple monophonic voice to very dense and polyphonic electronic music, without major failure. The operational range is also quite wide and there is no problem with changing the pitch by an octave. However, it suffers from several audio artifacts that makes the standard phase vocoder unusable for real musical applications.

Artifacts heard are chorus effect, transient smearing and phasiness (see section 2.4.2). These 3 artifacts are caused by the loss of vertical coherence in the phase vocoder. As explained previously, we use the instantaneous frequency at each frequency channel to recompute the new phase values at the corresponding frequency channel. But each frequency channel ignores its neighbouring frequency channels when computing these new values. This is a problem because, as explained in 3.2.1, a pure sinusoid in time domain is spread over several frequency channels in frequency domain. The phase vocoder then processes each frequency channel individually. The final result is not one pure pitch-shifted sinusoid, but a sum of several pitch-shifted sinusoids with very close frequencies. This lack of phase coherence between adjacent frequency channels explains the chorus effect and phasiness. On the other hand, the issue with transient smearing seems to be deeper. Horizontal phase propagation rely on having a stable periodic signal in each frame. What we see from each frame is a mean of all the frequency components inside the frame. However,
a transient only lasts a few milliseconds while the frame is 50 or 100 milliseconds long. These phase computations do not make sense for fast varying and short events, resulting in the smoothing of what is perceived as unexpected irregularities.

3.2.2 Phase-locked vocoder

Vertical phase coherence can be fixed if adjacent channels corresponding to the same input components are shifted "together" instead of individually. The first solutions to maintain vertical phase coherence were proposed in [20] and [21]. Phase values of the frequency channels which correspond to the same component are changed as a group. To make these groups, peak detection is made on the magnitude spectrum and frequency channels are assigned to their closest peak. Instantaneous frequency is computed for each peak and used to assign the new phase values for all its neighbouring channels. After implementing this solution, testing showed that it indeed reduced phasiness and chorus effect but that transient smearing was still a major issue.

Other solutions are proposed to maintain vertical phase. In [22], unlike standard phase vocoder implementation, horizontal coherence is voluntarily not perfectly maintained. By slightly translating the frames in time, phasiness effect can be better reduced. A similar idea is exposed in [23] where a phase vocoder implementation is combined with a synchronize overlap-add method, called PVSOLA.

3.2.3 "Phase vocoder done right"

Introduction to vertical phase propagation

In the classic phase vocoder implementation, the phase is only propagated in the time direction. To compute the phase of the frequency bin k at the current synthesis frame, we only look at the difference between the phase in the current and previous analysis frame for the bin k. In the phase-locked vocoder, the phase is locked around peaks in the frequency magnitude. The phase in the channel corresponding to the peak is propagated following the classic phase propagation, and the phase of the adjacent channels is computed using the same propagation factor as the peak bin. In the "phase vocoder done right" [24], the propagation can be both in the time or frequency direction. Depending on the magnitude on the different bins, the algorithm will choose whether it is suitable to propagate in the time or frequency direction. Conceptual differences between the 3 algorithms is illustrated on figure 3.11.
Figure 3.11: Conceptual difference between phase propagation in the standard phase vocoder, the phase-locked vocoder [20] and the phase vocoder done right from [24].
Phase computations and underlying idea

To compute the synthesis phase of the frame $n+1$, we need to know the magnitude and phase of the analysis frame $n+1$ and $n$ and of course the synthesis phase of the frame $n$. With this we can compute the phase time derivative $\Delta \phi_t(k, n+1)$, as for the standard phase vocoder, but also the phase frequency derivative $\Delta \phi_f(k, n+1)$ which you can compute by taking the phase difference between 2 adjacent frequency bins from the frame $n$.

These 2 derivatives are used to compute the synthesis phase in 3 different cases:

- if propagation in the time direction: $\phi_s(k, n+1) = \phi_s(k, n) + \Delta \phi_t(k, n+1) \beta$
- if propagation in the frequency direction (higher frequency): $\phi_s(k+1, n+1) = \phi_s(k, n+1) + \Delta \phi_f(k+1, n+1) \beta$
- if propagation in the frequency direction (lower frequency): $\phi_s(k-1, n+1) = \phi_s(k, n+1) + \Delta \phi_f(k-1, n+1) \beta$

The decision of whether the propagation should be in time or frequency direction depends on the bin magnitudes.

To understand why it is a fundamental improvement of the original phase propagation method, examples are shown on figure 3.12. First example corresponds to a sine wave centered on the frequency channel $m=3$. Horizontal phase propagation occurs along the frequency channel $m=3$, and then vertical phase propagation occurs from this frequency channel, ensuring the coherence between the adjacent channels without any explicit peak detection. Second example is a linear chirp with increasing frequency, third example is a sum of 2 sinusoids, fourth example is a transition from silence to impulse and fifth example is a transition from impulse to silence. Fourth example shows how this algorithm can reduce transient smearing. Phase propagation is only horizontal for one bin and then vertical for all the remaining bins. Most of the smoothing caused by horizontal phase propagation is reduced with this process.

More details on the algorithm can be found in appendix B. Results show that this algorithm greatly reduces transient smearing compared to the original phase vocoder. More details on the results are shown in section 4.2 where this algorithm is used in a polyvalent pitch-shifting implementation.

3.2.4 Transient preserving phase vocoders

Transient smearing is the major artifact created by frequency-domain transformation techniques. Recent focus on pitch-shifting has been done on improving transient processing. In [25], transients are explicitly detected and phase is reset at
transients to reduce transient smearing. In [26], frequency bin classification is done and different phase calculations depending on if the bin is classified as harmonic, noise or transient. In [27], harmonic and percussive signals are extracted from the input signal before the actual processing step. Harmonic signal is pitch-shifted using a phase vocoder and percussive signal using OLA. This reduces transient smearing as OLA is better at preserving transients. A transient preserving phase vocoder method inspired by this paper was implemented and is explained in section 4.2.

3.2.5 Multi-resolution phase vocoders

Wavelets vs Fourier

The Fourier Transform is the reference transform for time-frequency analysis. It is very computationally optimized but can suffer from a drawback in some contexts: its fixed time-frequency precision. Because of the uncertainty principle, it is not possible to localize perfectly a signal in both time and frequency domain. With Fourier analysis we can either:
- be precise in the time domain (beat tracking, transient detection...).
- be precise in the frequency domain (pitch detection...).

In the specific task of pitch-shifting. The choice is made to have a window which is roughly 100ms long in the time domain, which gives a frequency resolution of 10Hz in the frequency domain. The main issue is that we hear frequencies on a log
Figure 3.13: Comparison between STFT and CWT spectrogram on a drums track

Wavelets might be better suited for audio analysis because of their varying time-frequency resolution. High frequency components can be well localized in time while maintaining a good frequency resolution for low frequency components. A comparison between Wavelets (CWT) and Fourier (STFT) spectrograms is shown on figure 3.13. It can be seen that the frequency and time resolution is constant for all frequencies on the STFT spectrogram while it is varying on the CWT spectrogram.

Wavelets techniques in pitch-shifting

Some papers have developed the idea of pitch-shifting using multi-resolution time-frequency representation. The Continuous Wavelet Transform (CWT) does not seem
to be a very promising option. In [28], the CWT is used to pitch-shift speech. There is unfortunately not much information on the parameters of the CWT and no audio file is provided to assess its performance. Also the CWT can be very computationally expensive and would not be convenient in a real-time framework.

The Constant-Q Transform (CQT) is used for pitch-shifting in [29]. Compared to the CWT, the main advantage of the CQT is that the time step between 2 consecutive bins can be different for each frequency channel. In this paper, an octave-wise CQT representation is used, meaning that the time step is fixed for an octave and it is divided by 2 when we go to the lower octave. While a STFT representation is rectangular with a fixed frequency step and time step, this CQT uses a fixed frequency step in log scale, and a time step depending on the octave, as shown in figure 3.14. Phase propagation equations used in a CQT pitch-shifting are the same than for the original phase vocoder. However, the frame resampling step can be done much more efficiently than in the phase vocoder by shifting all the frequency channels up or down. Because a log scale is used for frequencies in the CQT, spectrum scaling or dilation in a linear scale is equivalent to shifting all the frequency channels up or down in a log frequency scale. If the CQT uses 48 channels per octave, a 1 octave up-shifting can be done by shifting all the bins by 48 frequency channels in the higher frequencies direction.

**Testing and conclusion on wavelets in pitch-shifting**

A CQT-based pitch-shifting algorithm was implemented in Matlab using the CQT toolbox from [30] and [31]. The non-fixed time-frequency grid proved to be the...
major problem when computing the phase propagation equation. Also, the use of a frame-by-frame implementation of the CQT [32] seems to cause reconstruction issues in an analysis-synthesis process. An offline CQT process was used instead and testing showed that the CQT performed slightly worse than the Fourier-based "phase vocoder done right" pitch-shifting algorithm.

In a task of general pitch-shifting, wavelets might not be suitable, or at least more research would need to be done in this topic because it is still a rather unknown and complicated area. Wavelets techniques would probably be the best option in the task of extracting and pitch-shifting individual notes from the signal due to their log frequency analysis scale.

3.3 Evaluation of pitch-shifting methods

From the preliminary study of pitch-shifting methods conducted in the previous sections, conclusions can be drawn on what could be potential applications of the described methods.

It appears that time domain methods are much simpler to understand and implement but they are quite limited in terms of applications. Time domain methods can be used with transposition factors very close to 1 to create a chorus effect. This application is implemented in section 4.1.5 TD-PSOLA can be used on monophonic signals and seems to be the best solution to make a vocals pitch-shifting, formant-shifting and pitch correction tool, implemented in section 4.1. Time domain methods are unable to pitch-shift complex polyphonic audio signals.

Phase-vocoder based methods are much more adaptive, as they can be used on any type of audio signal. They suffer from some artifacts, mostly because of transients and low frequency components, that can be reduced. Pitch-shifting range of phase vocoder methods is quite wide and quality is relatively well maintained for large factors. An implementation of the "phase vocoder done right" algorithm combined with formant-shifting and transient preservation is presented in 5.2. Major downsides of phase vocoder methods are their latency (as high as 150ms) and computational cost due to frequency analysis, transformations and synthesis.
Chapter 4

Applications

4.1 Voice correction, pitch and formant-shifting algorithm based on TD-PSOLA

This section details the design of a high quality pitch-shifting and formant-shifting algorithm for monophonic signals. A general pitch and formant-shifting application is designed as well as additional features derived from the general algorithm such as pitch correction and chorus effect. The core TD-PSOLA algorithm was described in section 3.1.2. This section is focused on the details and additional features added.

4.1.1 Pitch detection

There are 3 main steps in the TD-PSOLA algorithm: pitch detection, pitch marking and pitch-shifting. Pitch detection is the most critical part. In this section are presented some methods to obtain high quality pitch detection to be used in a TD-PSOLA pitch-shifting method.

FFT-based method

Similarly to the phase vocoder, instantaneous frequency is used to have a precise pitch estimate. Pitch is estimated every 256 or 512 samples. At each estimation step, DFTs of 2 consecutive frames are required. The hop size between these 2 frames can be as low as 1 sample and is different from the hop size of the pitch estimation which is 256 or 512 samples. The maxima of the magnitude spectrum are marked in the first frame. A magnitude threshold is used to keep only the relevant maxima. The remaining maxima of the magnitude spectrum are the pitch candidates. However, the frequency resolution of a 2048 samples DFT with a 44,1kHz sampling rate is 22Hz. This cannot give a precise estimate for the pitch. In the worst case, the real pitch falls exactly between 2 frequency bins and the pitch estimation...
To obtain a better estimate for the pitch, we measure the instantaneous frequency at pitch candidates. This is obtained by computing the unwrapped phase difference between 2 consecutive frames. This was explained in section 3.2.1. Difference between the uncorrected and corrected pitch estimates on a voice signal is shown on figure 4.1.

The pitch estimate precision was tested for pure sinusoidal signals. The typical error function relative to frequency is shown on figure 4.2, $f_{\text{min}} = f_s / N$ being the frequency resolution of the N points FFT in Hz, with $f_s$ the sampling-rate. Local maxima of the error are located in the middle of 2 frequency channels. Error is quite low for higher frequencies but can be an issue for low pitch detection.

**YIN and Spectral YIN**

YIN is a simple time domain method based on the auto-correlation function and providing good results compared to other pitch estimation techniques. It uses
Figure 4.2: FFT-based pitch estimation error relative to frequency

![FFT-based pitch estimation error relative to frequency](image)

the average magnitude difference function (AMDF) defined as:

\[
d_t(\tau) = \frac{1}{N} \sum_{k=t}^{t+N-1} (x[k] - x[k + \tau])^2
\]  

(4.1)

This function is computed at signal frames for all possible lags values \(\tau\). A normalized function is computed because the original AMDF is sensitive to amplitude changes. The normalized difference function (NMDF) is:

\[
d'_t(\tau) = \frac{1}{\tau} \frac{d_t(\tau)}{\sum_{k=1}^{\tau} d_t(k)}
\]  

(4.2)

Once the NMDF is computed, we iterate through the lags values. If the NMDF goes below a threshold (the dotted line on figure 4.3), the next local minima corresponds to the pitch estimate. Polynomial interpolation is used to compute a more precise estimate for the pitch. The absolute minimum cannot be taken as the best pitch candidate as it could correspond to an harmonic which is 2 times, 3 times or 4 times the true pitch value. An example of NMDF is presented on figure 4.3. Here, the first minima is located at the 420th sample corresponding to a pitch estimate of 105Hz.

A variation of the YIN method is spectral YIN [34]. It uses the tapered NMDF, a slightly different function than the original NMDF. It is defined by:

\[
d_t(\tau) = \frac{1}{N} \sum_{k=t}^{t+N-1-\tau} (x[k] - x[k + \tau])^2
\]  

(4.3)
CHAPTER 4. APPLICATIONS

Figure 4.3: Time signal and its NMDF

![Time signal and its NMDF](image)

\[
  d'_t(\tau) = \frac{d_t(\tau)}{\frac{1}{\tau} \sum_{k=1}^{N} d_t(k)} \quad (4.4)
\]

Only difference is that the number of terms in the AMDF sum depends on the lag value \( \tau \). This number decreases as \( \tau \) increases so the tapered NMDF tends to go to 0 for high lag values. This new function makes the pitch detection much simpler as the harmonics minima are noticeably higher than the fundamental minima and no threshold is required to detect the pitch value. The pitch estimate corresponds to the absolute minima of the tapered AMDF, as long a minimum lag value is fixed. Comparison between standard NMDF and tapered NMDF is shown on figure 4.4. The reason it is called spectral YIN is because this function is computed in the frequency domain. If \( x_i[k] \) is the input frame and \( X_i[k] \) its DFT, then the tapered AMDF computation in the frequency domain is:

\[
  d_t(\tau) = \frac{2}{N} \sum_{k=0}^{N-1} |X[k]|^2 \left(1 - \cos\left(\frac{2\pi k \tau}{N}\right)\right) \quad (4.5)
\]

The computational complexity is reduced to \( O(n \log(n)) \) as opposed \( O(n^2) \) in time domain [34], therefore spectral YIN is faster to compute. Relative error obtained with spectral YIN is consistently below 0.01Hz when testing with pure sinusoidal signals. For these reasons, spectral YIN is the preferred pitch detection method.
4.1.2 Pitch post-processing

The raw pitch estimate from the methods above are not sufficient to be used for TD-PSOLA so some post-processing steps need to be performed. It is mostly due to pitch detection in unvoiced parts (ex: ’s’, ’t’ or ’f’ sounds) which creates a very fast varying estimate and clicks in the audio signals. Some pitch estimation failures can also happen in voiced parts (ex: vowel sounds).

Voiced/Unvoiced detection

The following simple voiced/unvoiced detection process from [9] seems to be giving satisfactory results for TD-PSOLA. For each signal frame, the pitch is compared to the mean pitch of the k previous and k next frames. The delay is fixed to approximately 40ms and corresponds to k=7 frames with a 256 samples hop size for a sampling rate of 44.1kHz. As there is no precise pitch in unvoiced parts, the estimated pitch value varies a lot over time. The frame is labelled as unvoiced if the relative difference between the pitch and the mean pitch is higher than a a threshold. A 10% threshold seems to be giving the best results on most audio signals.

Most advanced techniques involving energy levels and zero-crossing rates were considered. They gave worse results and the optimal parameters varied a lot be-
tween tracks, even though these techniques give better classification on average. The reason is that it is better to classify an unvoiced frame as a voiced frame than a voiced frame as an unvoiced frame. Unvoiced frames do not require a specific window size, because they are either silences or short transients. Voiced frames require the window size to have a specific size depending on the pitch. If a voiced frame is detected as an unvoiced frame, the effect is a much more detrimental. The simple voiced/unvoiced detection described above gives a better classification on voiced frames, a worse classification on unvoiced frames but it is enough for this application.

Segmentation

With the previous step, we can obtain very short voiced or unvoiced segments, whose length can often be 1 frame (approximately 6ms). A voiced or unvoiced segment cannot be this short so these short segments have to be removed. This is the segmentation step. A minimum length for voiced/unvoiced segments is set to 40ms. First, short voiced segments are replaced by unvoiced segments and then short unvoiced segments are replaced by voiced segments.

Correction

After voiced/unvoiced detection and segmentation, the final step is to correct the pitch estimate with the segmentation obtained. Pitch on unvoiced segments is ignored and set to the pitch of the last voiced segment. To avoid pitch jumps and clipping at unvoiced to voiced transitions, the pitch of the unvoiced segment is "pre-loaded" to the next voiced segment pitch value if a voiced segment is detected in the next 40ms. Other post processing techniques involving pitch smoothing were tested but did not bring any relevant improvement.

Steps of pitch processing from raw pitch values to final values used for TD-PSOLA are shown on figure 4.5.

4.1.3 Formant-shifting

Voice transformation using TD-PSOLA was first developed in [35]. A simple similar formant-shifting method is used from [9]. As each frame extracted in TD-PSOLA corresponds to a filter model, the timbre can be changed by transforming the frames. We use frame resampling to increase or decrease the timbre of the voice or instrument. The timbre is lowered by increasing the width of the frame and increased by decreasing the width, see figure 4.6.
Figure 4.5: Pitch processing steps on a voice signal
In [36], parametric models are used in a TD-PSOLA framework in order to give a more natural timbre shifting. More complex and efficient formant-shifting methods based on spectral envelope estimation are presented in section 4.2.2.

4.1.4 Voice correction

General idea

Pitch detection is a necessary task when pitch-shifting with TD-PSOLA. The value of the pitch itself is not used in TD-PSOLA, but the pitchmarks positioning is based on the detected pitch. A voice correction feature can be added very easily to this pitch-shifting implementation. Depending on the key a song is written in, some notes are or are not supposed to happen. For example, if a song is written in a C minor key, there should not be an E note. Instead of using a fixed pitch-shifting factor $\beta$, a variable factor is used so that the pitch of each frame is set to the closest "true" tone on a musical scale.

Example:
- pitch of B2 note is 123.4Hz
- pitch of the C3 note is 130.8Hz detected input pitch is 128Hz. Its closest correct note is C3. In order to shift the input to C3, the pitch-shifting factor must be $\beta = 130.8/128 = 1.021$.

In this naïve method, the correction is instant and no smoothing of the pitch is done over time, resulting in the (in)famous "Autotune" effect. Autotune can be heard as non natural singing, with the pitch instantly set to certain values. Such effect can be desirable or not depending on the case. Plots of input and autotuned pitch are shown on figure 4.7. It can be seen that the Autotune removes the vibrato effect of the singer at 1.5s or 5.5s.
Figure 4.7: Input pitch and corrected pitch of an extract from Bohemian Rhapsody, without any smoothing

**Correction smoothing**

To provide more natural voice correction, smoothing is necessary. Smoothing is applied to the pitch-shifting factor $\beta$ with an IIR filter. If $\beta_s[n]$ is the smoothed pitch-shifting factor of the $n^{th}$ frame, and $\beta[n]$ the unprocessed pitch-shifting factor of the $n^{th}$ frame, then:

$$\beta_s[n] = \beta[n]q + \beta_s[n-1](1-q), \hspace{1em} 0 \leq q \leq 1 \hspace{1em} (4.6)$$

With $f_s$ the sampling-rate in Hz, $h_s$ the step in samples between 2 pitch measurements and $t_r$ the time in seconds it requires for the smoothed pitch-shifted to exceed 50% of the final value in response to a Heaviside step $u[n]$, $q$ is defined as follows:

$$q = 1 - 0.5 \frac{n_r}{n_r+1}, \hspace{1em} n_r = \frac{f_s}{t_r h_s} \hspace{1em} (4.7)$$

Details are left in appendix C. $t_r$ is the user-friendly parameter that is let to choose pitch correction speed. A value of 50ms gives a rather natural pitch correction while autotune effects can be clearly heard below 20ms. Difference between processed and unprocessed pitch-shifting factor is shown in 4.8 and plots of input and smoothed autotuned pitch are shown on figure 4.9.

**4.1.5 Chorus effect**

From [9], it is described that a chorus effect can be obtained by adding several versions of the same signal but with slightly different pitches. Historically, this was
Figure 4.8: Comparison between unprocessed and smoothed pitch-shifting factor over time

![Comparison between unprocessed and smoothed pitch-shifting factor over time](image)

Figure 4.9: Input pitch and corrected pitch of an extract from Bohemian Rhapsody, with smoothing

![Input and corrected pitch of a voice signal with correction smoothing](image)
done by using delay line modulation \cite{delay_line_modulation}, an older pitch-shifting technique used for the first digital pitch-shifting hardware in the 1970s \cite{pitch_shifting_hardware_1970s}. Results can be obtained with any pitch-shifting method. Here, we pitch-shift an input signal with TD-PSOLA, using different pitch-shifting factors and summing the pitch-shifted signals.

A mono signal is taken as an input. The output signal is a stereo signal. The original signal is equally added to the left and right outputs. Then, slightly up-shifted versions of the signal are added to the left output, and slightly down-shifted versions of the signal are added to the right output. This creates a stereo and chorus effect on the output. More stereo effects can be obtained by choosing wisely the positioning of the shifted signals on left and right outputs but it would be beyond the scope of this project.

4.1.6 Implementation

Implementation of the algorithm was done using Jupyter Notebook and a Graphical User Interface (GUI) was made. Chorus effect is separated from the pitch-shifting, formant-shifting and autotune tools. Overview of the GUI is presented in appendix \ref{appendix:gui}. The process is offline: the user has to import a wavfile, start the process and listen to the output file once processing is finished. The pitch-shifting/autotune tool is quite fast and takes less than 2s to pitch-shift a 15s input signal. Chorus effect is still to be optimized and would be almost as fast as the classic pitch-shifting tool if code is refined.

4.1.7 Results

Generated audio files are presented in \cite{generated_audio_files}. Headphones are very much advised when listening to the results. Pitch-shifting and formant-shifting results can be listened to in "TD-PSOLA - Pitch-shifting". The cleanest results are obtained when the pitch and timbre are the same (+4,+4 or -4,-4), but they alter the timbre of the voice. Down-shifting without changing the timbre is what creates the worst results. Low frequency modulation can be heard and gives a robotic timbre to the voice. Up-shifting without changing the timbre works fine as long as reasonable pitch-shifting factors are used. TD-PSOLA still works when up-shifting by an octave but it sounds much less natural. Changing the timbre without changing the pitch gives a natural result when the timbre is decreased. This is not the case when increasing the timbre, as it gives a very nasal voice.

In "TD-PSOLA - Autotune" are presented a voice track and several autotuned versions of it using different correction speeds. Depending on the correction speed chosen, we can either obtain the autotune effect (instant correction), have a more natural yet very corrected voice effect (10ms and 20ms correction) or a more subtle
pitch correction (30ms and 50ms correction).

Chorus effect results are presented in "TD-PSOLA - Chorus Effect". The number of voices when using a low maximum pitch-shifting factor (-0.06 semitones to +0.06 semitones) does not seem to be very important. In both cases, a good quality chorus effect is generated. When using larger maximum pitch-shifting factors (-0.3 semitone to +0.3 semitone), the number of voices has a significant impact on the result obtained. Detuning of the additional voices is heard when there are only 2 of them, while a more coherent set of voices is obtained when using 10 additional voices. A wide range of chorus effect can be achieved by using different sets of parameters.

4.2 Polyvalent transient preserving pitch-shifting and formant-shifting algorithm based on the phase vocoder

4.2.1 Pitch-shifting method

The phase vocoder algorithm used is the "phase vocoder done right" [24] presented in section 3.2.3. Massive improvement of this algorithm compared to the standard phase vocoder comes from its time and frequency phase propagation pattern. This greatly reduces transient smearing on the phase vocoder, as shown on figure 4.10.

A fix of the "phase vocoder done right" was implemented. When down-shifting, high frequency noise (above 15kHz) could be heard. The reason for this high frequency noise is still unclear. The solution found to solve the problem was to oversample the frames by a factor 2 in the frequency domain. In practice, this was done by zero-padding the DFT coefficients. First 4096 bins are computed from the DFT and 4096 zeros are added. Because the frame length is doubled in the frequency domain, the length is also doubled in time domain. To go back to the original length, down-sampling by a factor 2 has to be computed at frame reconstruction. Waveform and spectrum magnitude differences when using oversampling or not using oversampling are shown on figure 4.11. An increase in the spectrum magnitude at 15kHz as well as a high frequency modulating noise on the waveform can be observed, representing the high frequency noise.

Even though the "phase vocoder done right" reduces transient smearing, pre-echo artifact (see section 2.4.2) can still be heard on drums tracks. It can be seen on figure 4.12 as the little oscillations before the transient. A pre-echo reduction step is designed and explained in appendix F.
Figure 4.10: Waveforms of a transient pitch-shifted with standard phase vocoder and "phase vocoder done right"
Figure 4.11: Oversampling effect on high frequency noise in the "phase vocoder done right"
4.2.2 Formant-shifting with spectral envelope estimation

General idea

One negative property of the phase vocoder is that it does not preserve formants. While TD-PSOLA naturally preserves formants, the phase vocoder naturally shifts formants. To preserve formants in the phase vocoder, formant-shifting has to be added as an extra feature.

Formant-shifting is based on spectral envelope (see figure 2.6) estimation. By estimating the envelope of the input frame, we can know beforehand the envelope of the pitch-shifted frame. If we define $A(f)$ the estimated envelope, $\beta$ the pitch-shifting factor, then the pitch-shifted envelope is $A_s(f) = A(f/\beta)$. To compensate the envelope shifting, we multiply the frame magnitude by a correction envelope $C(f)$ defined by:

$$C(f) = \frac{A(f)}{A_s(f)} = \frac{A(f)}{A(f/\beta)} \quad (4.8)$$

An example is illustrated on figure 4.13. The hard part of the work consists in estimating the envelope of the input frame. Several methods are presented in the next sections.
Cepstrum envelope estimation

Cepstrum analysis is the basic method for envelope estimation. The log magnitude of the DFT is computed. This gives the spectrum magnitude on a logarithmic scale. Envelope is computed by applying a low pass filter on the log spectrum. This is done in the cepstrum domain by computing the iDFT, multiplying the cepstrum coefficients by a window whose size is defined by the cepstrum order, and then going back to the frequency domain by computing the DFT.

Doing so, the cepstrum envelope estimate is obtained. Essentially, this envelope is a moving average of the spectrum and because of that, it can miss a lot of information. This is illustrated on figure 4.14 where the spectrum of a frame is plotted alongside another envelope estimate. The cepstrum envelope does not follow the peaks of the spectrum and mismatches the "correct" envelope especially in the low frequencies.

Peak-based envelope estimation

A novel method using peak detection of spectrum magnitude was developed to estimate the envelope. After computing the log magnitude of a frame, we iterate through the DFT bins. To be considered as a local maximum, the magnitude of the DFT bin needs to be higher than its neighbours magnitude in a certain frequency range. The choice of the correct range is crucial because the algorithm might add non relevant peaks or forget relevant peaks. Depending on the fundamental frequency, the step between harmonics is not the same, so this range cannot be fixed. The iteration starts at the 8th frequency bin. For a 4096 samples window at 44.1kHz it corresponds to 86Hz. Formants being mostly relevant for voice and chord instruments, the peaks should not be detected below this frequency because the pitch should not be that low. But the main reason for ignoring the first DFT bins is that the precision in semitones below 100Hz is quite low. In our case, the frequency resolution is 10.7Hz, and this corresponds to approximately 2 semitones at 80Hz. Matching the imprecise envelope to the DFT at these frequency brings some problems such as heavy low frequency amplification. The range is set to a default minimum value (15 bins, ≈160Hz). Then, once the first peak is detected, the value of the range is computed based on the first peak so that it matches the detected fundamental frequency (higher range if higher fundamental frequency, lower range if lower fundamental frequency). Once the range is set, we iterate through the DFT bins and look if the bin has the maximum magnitude in its neighboring range. If so, we mark the bin and jump to the right edge of the range because no other peak can be found in the current range.
Figure 4.13: Computation of correction envelope based on input frame envelope estimation
I added some features in case the first peak detected is not the fundamental. First case, the detected first peak is a sub-harmonic (1/2 of the fundamental frequency). The search range is narrower than it should and it results in over-fitting the spectral envelope. This adds a heavy noisy chorus effect. To prevent sub-harmonic detection, the amplitude and frequencies of the first 2 peaks are compared. If the frequency of the 2nd is roughly twice the frequency of the first and the magnitude of the 1st is much lower than the magnitude of the 2nd, then the first peak is deleted and the process redone with the search range being twice as large this time. Second case, the first peak is an harmonic above the fundamental. The search range is too large and we miss harmonics in the envelope estimation. After computing the peaks, we check if the distances between them are not too large. If a distance between 2 peaks is suspiciously large, then we fill in the blank by adding a “peak” in the middle of the 2 adjacent peaks. This feature is also useful even when the search range is correct but when the envelope has a concave shape. This is illustrated on figure 4.15. The magnitude of the FFT is represented by the blue line, a good spectral envelope by the red line, and the detected peak in the actual spectral envelope detection by the red dots. When not filling the blanks, this algorithm ignores the concave part of the envelope. Thanks to this feature, it adds the green dot and gives a correct envelope. This makes a very significant difference in the resulting audio quality.

The envelope is then constructed by using linear interpolation between the peaks. Using 2nd order polynomial makes the envelope look smoother but does not improve the audio results. Main issue of this method is that it is built in a very empirical way. It was tested on different signals and features were iteratively added to remove the
failures one by one until it worked fine on these test signals. So there could be some unencountered cases where it can fail. Example of the peak-based estimated envelope is shown on figure 4.14.

True envelope estimation

The true envelope estimation is described in [39]. The issue with the standard cepstrum envelope is that it follows the general trend of the envelope but it is below the magnitude peaks. This is solved by building iteratively an envelope from the cepstrum envelope to fill the parts of the spectrum where the envelope is below the peaks.

At the first iteration, the cepstrum envelope is computed from the original spectrum as described in section 4.2.2. Then, for each frequency bin, the envelope value is replaced by the maximum between the envelope value and the original spectrum value. At the next iteration, the updated envelope is used instead of the original cepstrum envelope. Doing so, the valleys of the spectrum are progressively filled until the correct envelope is obtained. The first step is depicted in figure 4.16. In [39], some tricks are presented to speed up the iterations and the authors manage to obtain the envelope estimate within 5 to 10 iterations. However, some issues with the original algorithm had to be fixed:

- the envelope was not smoothed enough.
- the envelope was way above the DFT magnitude at very high frequencies, adding noise.
- the envelope was inaccurate at low frequencies.

To smooth the envelope, a median filter is applied. It makes the envelope perfectly usable in a formant preservation/shifting application. To remove the high frequency noise, the envelope is set to a constant value at the last frequency bins. To
compensate the bad accuracy at low frequency, a solution is to ignore the correction between the first bin and the first peak bin by setting the correction envelope value to 1. An example of final envelope estimate can be seen on figure 4.17. Because of its high reliability, this method was preferred over the peak-based estimation method.

4.2.3 Transient preservation

This section presents a harmonic-percussive separation method which improves transient processing in the phase vocoder. The motivation behind the use of a harmonic-percussive separation method is that the transient resonance solely defines its pitch. Therefore we should not change the attack of a transient when pitch-shifting it but only the resonance. The "phase vocoder done right" manages to maintain a decent quality when upshifting or slightly downshifting transients. However, it becomes clear that the transient attack are badly affected when using larger downshifting factors. The attack becomes very weak and this can be heard on
Figure 4.17: True envelope estimation method

Both the phase vocoder and also the reference commercial pitch-shifting algorithm elastiquePro.

Once the percussive component is separated from the harmonic content (figure 4.18), the harmonic component is pitch-shifted using the "phase vocoder done right" algorithm. After that, the percussive component is added to the pitch-shifted harmonic component. The idea and steps of the method are mainly inspired from [27], but with differences in both the separation step and pitch-shifting step. The separation method itself is more deeply explained and illustrated in appendix E. The pitch-shifting step is also quite different because they use OLA to pitch-shift the percussive component while we decide to keep the percussive component as it is. The motivation behind this choice is that the percussive component should not be changed and, even if we wanted to change the pitch of the attack, OLA is a very weak method which can bring a lot of artifacts. Using a phase vocoder algorithm with a shorter window for percussive components would probably be a better idea in that case.

The harmonic-percussive separation method massively reduces transient smearing when down-shifting. Comparison between a drum double clap, its down-shifted version with this method and its down-shifted version using the commercial algorithm elastiquePro is depicted on figure 4.19. On the figure, the claps are the fast varying bursts of energy located at 0.01 and 0.04s. The down-shifted claps with this method are identical to the input clap at the attacks, and the difference in pitch can be observed at the resonance. The first and second down-shifted clap with the commercial algorithm elastiquePro are more spread in time and the second one
has a much lower amplitude. The difference is less obvious when up-shifting but harmonic-percussive separation is still better as preserving transient attacks.

4.2.4 Implementation

Implementation of these algorithms was done using Jupyter Notebook in 2 different modules: the "phase vocoder done right" implementation with formant-shifting, and the harmonic-percussive separation pitch-shifting algorithm. Overview of the GUI is presented in appendix D. "Phase vocoder done right" implementation is still quite slow in Python, approximately 3 times as slow as real-time, 10s of audio are processed in 30s. An executable provided by the author of the article runs in real-time with reasonable CPU usage so it is reassuring concerning the possibility to implement this algorithm in a real application. Formant-shifting is also quite slow at the moment and can be sped up following the tricks described in [39].

4.2.5 Results

Generated audio files are presented in [38]. Comparison is made with the polyvalent commercial algorithm elastiquePro. This algorithm is implemented in various digital audio workstations (DAW) and is the reference tool for pitch-shifting any audio signal.

The results of the "phase vocoder done right" implementation are presented in "Phase Vocoder - Pitch-shifting". On pure instrumental tracks, both algorithms are very comparable. It is difficult to identify each algorithm in a blind-test. However, the distinction is not so subtle on voice or when using extreme pitch-shifting factors.
Differences on voice pitch-shifting is presented in the results. ElastiquePro creates a more dynamic sound but less natural timbre than the "phase vocoder done right". With extreme-pitch factors, differences between the 2 algorithms are amplified. It is still hard to determine which gives the best results because using such extreme transposition factors, the results cannot sound good or natural and it is very track dependent.

The results of the harmonic-percussive separation upgrade are shown in "Phase Vocoder - Transient preservation". Only down-shifting factors are used in the examples because the difference when up-shifting is slighter, but can still be heard. Each transient attack is more preserved and sharper when using transient preservation. The most obvious example is the dubstep snare downshifted by 2 octaves. ElastiquePro flattens the attack while it is preserved when using the algorithm we presented. Similar results can be heard on the other complete drums examples with more reasonable pitch-shifting factors (-6 to -10 semitones).
Chapter 5

Conclusions and future work

5.1 Conclusions

Two different offline pitch-shifting algorithms have been implemented based on an extensive literature study and novel features. A time domain algorithm, TD-PSOLA, is used for various effective voice transformations. It is able to perform pitch and formant shifting, pitch correction and chorus effect with promising quality. A frequency domain algorithm, the phase vocoder, is able to do time-frequency transformations of complex audio signals. Testing shows that this implementation is comparable to the reference commercial pitch-shifting algorithm elastiquePro in terms of audio quality on most content. A novel harmonic-percussive separation was used to improve drums pitch-shifting and provides unique audio quality. Audio files are presented in [38].

5.2 Future work

Deeper research could be made on some aspects of the presented pitch-shifting algorithms.

Code optimization and real-time implementation of the "phase vocoder done right" algorithm in C++ would give a better idea on whether this algorithm is suitable or too computationally consuming for a real application. Optimization of the spectral envelope estimate for formant preservation could be done to reduce the computation time and improve envelope estimation. Percussive and harmonic separation process should be improved to be more consistent.

Real-time Constant-Q transform analysis-synthesis scheme could be studied to open pitch-shifting to the wavelets domain. No framework to use wavelets transforms in a real-time framework for pitch-shifting was found in the literature for now. It is assumed that some sophisticated plugins already use multiresolution time-frequency representations.
Bibliography


[34] Paul Brossier. “Automatic annotation of musical audio for interactive applications”. In: 2006.


[38] Pitch-shifting sound files. URL: https://sites.google.com/eiosis.com/pitch-shifting.

Appendix A

Amplitude flatness

Examples of amplitude flatness (AF) in function of the overlap ratio from [7] are presented in figure A for the Hamming and Kaiser windows. In simple OLA method, the Kaiser window is used with an overlap of 75%. When down-shifting, the overlap is decreased which results in a decrease in amplitude flatness and a modulation effect. In PSOLA, the Hanning window is used with 50% overlap. When down-shifting (reducing the overlap) the amplitude flatness quickly drops and creates the modulation effect. When up-shifting, the amplitude flatness is less affected and stays close to 1.
Figure A.1: Amplitude flatness (pink) in function of overlap ratio for Hanning and Kaiser windows
Appendix B

Details on the ”phase vocoder done right” algorithm

B.1 Algorithm

The algorithm uses a max heap data structure. Elements can be pushed into a max heap and the element popped is always the maximum value of the heap. Example: If we add 4, 1, 6 to an empty heap and pop 2 elements, we obtain 6 and 4. In our case, what we push into the heap is a tuple whose 1st element is the magnitude of the bin, the 2nd is the frequency index, the 3rd is the time index. Example: If the magnitude of the frame 2 at the frequency bin 6 is 0.3, we push a tuple (0.3,6,2) into the heap.

The following algorithm should be iterated to compute the phase of each synthesis frame n+1. Some pre-processing steps need to be done first: compute the phase time and frequency derivatives and assign random phase values if some frequency bins in frame n+1 have a very small magnitude (below a threshold). Now we can push into the heap all the bins of the frame n where the magnitude is above the threshold. At the first iteration, we pop one element from the heap. It is the bin with the highest magnitude and it has to be in the frame n because at this moment, all the elements of the heap come from the frame n. If the extracted element is \((G_h, k_h, n_h)\) where \(G_h\) is the magnitude of the bin, \(k_h\) its frequency index and \(n_h\) its time index, we look at the bin from the same frequency channel but at the next frame \((G'_h, k_h, n_h + 1)\). If the magnitude \(G'_h\) is above the threshold, then we propagate the phase in the time direction. Once we have computed the synthesis phase, we add this new element to the heap. If the magnitude \(G'_h\) is below the threshold, it means we have already assigned to this bin a random phase value, so we can go to the next element of the heap.
For the 2nd iteration, we have 2 options: the element popped from the heap belongs to the frame n-1 or the element popped from the heap is the element from the frame n we just added to the heap. In the first case, we just repeat the first iteration and compute the time propagation if the magnitude of the new element is above the threshold. In the second case, if the extracted element is \((G_h, k_h, n_h)\), we will look at the elements \((G''_h, k_h + 1, n_h)\) and \((G'''_h, k_h - 1, n_h)\) which are the bins at the same frame but in the adjacent frequency channels. If the magnitudes of the new elements are above the threshold, we propagate the phase in the frequency direction. Then we add the new elements to the heap and repeat... Once all the synthesis bins have been “visited” and phase-propagated, it stops. The algorithm ensures that there is no overwrite for the synthesis phase values.

### B.2 A simple example

Each column corresponds to a signal frame (time axis), each row corresponds to a frequency channel. Each number inside of a bin is the magnitude of the frequency bin at a given frame. The phase of the current frame (2nd column) needs to be computed from the last frame (1st column).

![Table](image)

The bins of the last frame are put in the heap (yellow)
The max of the heap (30) is extracted (red) and the phase is propagated in the time direction as the max of the heap corresponds to a bin from the last frame. The new bin is added to the heap.

The new max is still in the last frame so it is again a time propagation. The new bin is added to the heap.

Now the max of the heap is a bin from the current frame. The phase is propa-
gated in the frequency direction. Phase is propagated to the higher frequency bin, but the lower frequency bin has already been computed at the last step. The new bin is added to the heap.

\[
\begin{array}{ccc}
-30 & 30 & 20 \\
30 & 20 & 10 \\
25 & 10 & -30 \\
-30 & -30 & -30 \\
\end{array}
\]

The maximum is extracted for the heap, it is a bin from the current frame but there is no option to propagate in the frequency direction so nothing happens.

\[
\begin{array}{ccc}
-30 & 30 & 20 \\
30 & 20 & 10 \\
25 & 10 & -30 \\
-30 & -30 & -30 \\
\end{array}
\]

The last propagation for the current frame is computed. Then it switches to the next frame. This process is illustrated below for the next frame. It was assumed for this example that for equal magnitude values, the bin from the last frame is always preferred.
APPENDIX B. DETAILS ON THE "PHASE VOCODER DONE RIGHT" ALGORITHM

<table>
<thead>
<tr>
<th>-30</th>
<th>30</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>25</td>
<td>10</td>
<td>-30</td>
</tr>
<tr>
<td>-30</td>
<td>-30</td>
<td>-30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>-30</th>
<th>30</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>25</td>
<td>10</td>
<td>-30</td>
</tr>
<tr>
<td>-30</td>
<td>-30</td>
<td>-30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>-30</th>
<th>30</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>25</td>
<td>10</td>
<td>-30</td>
</tr>
<tr>
<td>-30</td>
<td>-30</td>
<td>-30</td>
</tr>
</tbody>
</table>
### APPENDIX B. DETAILS ON THE “PHASE VOCODER DONE RIGHT” ALGORITHM

#### Table 1: Phase Values

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-30</td>
<td>30</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>20</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>10</td>
<td>-30</td>
<td></td>
</tr>
<tr>
<td>-30</td>
<td>-30</td>
<td>-30</td>
<td></td>
</tr>
</tbody>
</table>

#### Table 2: Phase Values

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-30</td>
<td>30</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>20</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>10</td>
<td>-30</td>
<td></td>
</tr>
<tr>
<td>-30</td>
<td>-30</td>
<td>-30</td>
<td></td>
</tr>
</tbody>
</table>

#### Table 3: Phase Values

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-30</td>
<td>30</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>20</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>10</td>
<td>-30</td>
<td></td>
</tr>
<tr>
<td>-30</td>
<td>-30</td>
<td>-30</td>
<td></td>
</tr>
</tbody>
</table>
### APPENDIX B. DETAILS ON THE "PHASE Vocoder Done Right" Algorithm

<table>
<thead>
<tr>
<th></th>
<th>-30</th>
<th>30</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>20</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>10</td>
<td>-30</td>
<td></td>
</tr>
<tr>
<td>-30</td>
<td>-30</td>
<td>-30</td>
<td></td>
</tr>
</tbody>
</table>
Appendix C

Details on pitch correction smoothing

In this section, details on the computation of the smoothing filter in pitch detection are given. If the $\beta_s[n]$ is the smoothed transposition factor and $\beta[n]$ the unprocessed transposition factor, the smoothing filter used for pitch correction is defined by:

$$\beta_s[n] = \beta[n]q + \beta_s[n-1](1-q), \ 0 \leq q \leq 1 \quad (C.1)$$

We can compute the transfer function $H(z)$ of this filter:

$$H(z) = \frac{\beta_s[z]}{\beta[z]} = \frac{q}{1 - (1-q)z^{-1}} \quad (C.2)$$

We want to control the time it requires for the smoothed transposition factor to exceed 50% of its final value in response to a Heaviside step $u[n]$. The Z-transform of $u[n]$ is $\frac{1}{1-z^{-1}}$. The Z-transform of the step response is defined as follows:

$$\beta_s[z] = \frac{q}{1 - (1-q)z^{-1}} \frac{1}{1-z^{-1}}$$

$$= \frac{q - 1}{1 - (1-q)z^{-1}} + \frac{1}{1-z^{-1}} \quad (C.3)$$

By computing its inverse Z-transform, we obtain the step response in time domain:

$$\beta_s[n] = (q - 1)(1-q)^n u[n] + u[n]$$

$$= u[n](1 - (1-q)^{n+1}) \quad (C.4)$$

The parameter $q$ is defined by the pitch frame index $n_r$ at which we exceed 50% of the final value:

$$1 - (1-q)^{n+1} = 0.5$$

$$q = 1 - 0.5^{\frac{1}{n+1}} \quad (C.5)$$
$n_r$ can be replaced by $t_r$, a more user-friendly parameter whose response time in given in s. With $f_s$ the sampling-rate in Hz and $h_s$ the step in samples between 2 pitch measurements, relation between the 2 parameters is given by:

$$n_r = \frac{f_s}{t_r \cdot h_s}$$

(C.6)
Appendix D

Graphical User Interface of pitch-shifting algorithms

Figure D.1: GUI of pitch-shifting/Autotune voice tool

Figure D.2: GUI of voice chorus tool
### Figure D.3: GUI of phase vocoder pitch-shifting tool

<table>
<thead>
<tr>
<th>Start (s)</th>
<th>End (s)</th>
<th>Pitch (ST)</th>
<th>Timbre (ST)</th>
<th>Win width</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20</td>
</tr>
</tbody>
</table>

- Locked Pitch/Timbre
- Pre-echo Reduction
- Force mono output

- Start Processing
- Select Files

### Figure D.4: GUI of transient preserving pitch-shifting tool

<table>
<thead>
<tr>
<th>Start (s)</th>
<th>End (s)</th>
<th>Pitch (ST)</th>
<th>Timbre (ST)</th>
<th>Win width</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20</td>
</tr>
</tbody>
</table>

- Locked Pitch/Timbre
- Pre-echo Reduction

<table>
<thead>
<tr>
<th>Win Sop</th>
<th>Hop Sop</th>
<th>Median T</th>
<th>Median f</th>
<th>diffThres(dB)</th>
<th>eThres(dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>128</td>
<td>15</td>
<td>11</td>
<td>5</td>
<td>-80</td>
</tr>
</tbody>
</table>

- Plot figures
- Reset Param
- Start Processing
- Select Files
Appendix E

Harmonic-Percussive separation algorithm

In this section is described step by step how the percussive and harmonic components of the signal are separated using a novel technique, inspired from [27].

The STFT of the input signal is computed with a window size of 1024 samples and a hop size of 128 samples. This gives the following spectrogram.

Then, we apply median filters to the spectrogram. Using a median filter along the frequency axis, we obtain the percussive spectrogram. Using a median filter along the time axis, we obtain the harmonic spectrogram.
What can be seen is that the median-filtered bins along the frequency axis have a higher amplitude than the median-filter bins along the time axis at transient attacks. Similarly, during resonances, the median-filtered bins along the time axis have a higher amplitude.

We build a percussive mask which is 1 when the magnitude of the percussive spectrogram is a certain amount of dB higher than the magnitude of the harmonic spectrogram, and 0 otherwise. All the previous steps were directly taken from [27]. This step differs a little because they use a fixed threshold value of 0 dB while it was decided to use a threshold as a parameter. The next steps are completely new. The percussive mask is shown below.
This mask is still very noisy and cannot be used yet. To make things easier, we classify an entire time frame as either percussive or harmonic, which means all the bins at a time index correspond to the same component. To filter this mask, a technique which is typically used in image processing was experimented: morphological erosion. First we want to remove the isolated frequency bins in resonances by doing a erosion along the frequency axis.

There is still some isolated bins so we use erosion along the time axis to remove them.
Now, as we want all the frequency bins of the same frame to be identically classified, if one mask frequency bin is 1, the whole frame is considered as percussive.

We now multiply the spectrogram by this mask and the inverse mask to obtain the percussive and the harmonic spectrogram.
Finally, iSTFT is used to reconstruct the 2 signals. Below is plotted the extracted percussive signal overlapped to the original signal.
APPENDIX E. HARMONIC-PERCUSSIVE SEPARATION ALGORITHM

Percussive-Harmonic separation of a drums signal

![Graph showing the separation of percussive and harmonic signals from audio samples.](image-url)
Appendix F

Pre-echo reduction processing

Pre-echo reduction is a feature added to the phase vocoder to remove noisy chirp which can be heard before transients. Its functioning is described in figure F.1. The mean absolute value of the input and pitch-shifted signals are computed on 3ms windows. These gains are computed on a log scale instead for simplicity. Smoothed gains of the input and pitch-shifted are obtained doing so. Then, a gain reduction signal is obtained by subtracting the pitch-shifted gain to the input gain (in dB). The gain reduction signal describes the difference in amplitude level between the input and pitch-shifted signal. If there is pre-echo on the pitch-shifted signal and nothing on the input signal, the gain reduction is very low (≤-100dB). During the transient, gain reduction should be close to 0 dB because input and pitch-shifted signals should have the same amplitude. A thresholding step is added so that gain reduction is only applied for pre-echo reduction. To do so, a threshold is used on the value of the input gain. If the input level is low (≤-70dB for example), there could be pre-echo on the output so the gain reduction is not changed. If the input level is high (≥-40dB for example), then we should not affect the output signal and the gain reduction is set to 0dB. Between these 2 threshold values, the gain reduction is linearly scaled so that gain reduction is not affected at -70dB and completely ignored at -40dB. The gain reduction signal is finally computed back from a log scale to a linear scale and multiplied to the pitch-shifted signal.
Figure F.1: Block diagram of pre-echo reduction processing