Polyphonic Pitch Modification Using Phase Vocoder Techniques

Erik Strandh

Thesis for a diploma in computer science, 30 ECTS credits
Department of computer science, Faculty of science, Lund University

Examensarbete för 30 hp
Institutionen för datavetenskap, Naturvetenskapliga fakulteten, Lunds universitet

Polyphonic Pitch Modification Using Phase Vocoder Techniques

Abstract

In the field of digital music processing, the phase vocoder is a well-established tool for pitch shifting and time scaling. Dolson and Laroche (1999) suggest a peak based phase vocoder, which opens the door for a much wider range of spectral modifications. We investigate the ability to change single notes in polyphonic music recordings using this technique, and the underlying theory is reviewed to provide motivation for the main implementation choices. We found that good results can be obtained for shifts of one or a few semitones, and possibly for even greater shifts, though at a great computational cost. Finally, we discuss the possibility of real time application and simple refinements to obtain other effects.

Polyfon tonjustering baserad på en fas-vocoder

Sammanfattning

Fas-vocodern ett väletablerat verktyg för att ändra ton och tempo unisont i digitala musikinspelningar. Dolson and Laroche (1999) förslår en peakbaserad fas-vocoder, som möjliggör en mängd andra operationer på spektrat. Vi undersöker möjligheten att ändra enskilda noter i polyfona musikinspelningar med hjälp av denna teknik, och går igenom den underliggande teorin för att motivera de viktigaste implementationsbesluten. Vi har funnit att goda resultat kan uppnås för ändringar på några få halvtoner, och möjligen för större ändringar, men då mot en hög beräkningskostnad. Slutligen diskuterar vi möjligheten till realtidstillämpning, samt några enkla förbättringar som kan leda till större användbarhet.

Contents

Chapter 1

Introduction

1.1 Overview

A digitally sampled sound is a sequence of discrete values representing the pressure of the sound wave at discrete points in time. There are many techniques for analyzing and modifying digital audio signals, which is a large subfield of digital signal processing. Some of the most powerful techniques relies on Fourier analysis to transform the signal from the time domain to the frequency domain.

For the purpose of music production, some operations on the frequency domain representation of a signal open the door for a great range of desirable musical modifications. Probably the most important of these modifications is that of changing the pitch of the notes in a recording without changing the duration, *pitch-shifting*. This is closely related to the ability to change the duration without changing the pitch, *time-stretching*, which is equally useful.

To clarify this, picture a vinyl record that is played at twice the intended speed. The result will be music that is one octave higher than the original – corresponding to doubling the frequencies – and the duration will, of course, be half of the original. The digital counterpart of the vinyl player example is a *down sampling* of the signal, i.e. removing every second sample point from the sample sequence. The result is virtually the same; the sound will be played at twice the speed.

The conclusion is that once the pitch has been changed without changing the duration, it is a relatively simple matter of sample rate conversion to restore the pitch with a change in duration, or vice versa.

A common approach to time-stretching/pitch-shifting is the *short-time Fourier transform* (STFT). The basic principle is to produce spectra from short segments of the signal using the *discrete Fourier transform* (DFT). One can then modify each spectrum according to the desired result, before applying the inverse of the transformation and recombining the modified segments.

When modifying the spectra with the intention to generate a different sound, it is important that the phase of the constituent sinusoids is taken into consideration. This brings us to the concept of the *phase-vocoder*, which uses the STFT to produce a time-varying spectrum, or *spectrogram*, and applies modifications to its spectra in a way that preserves phase coherence between adjacent segments.

Dolson and Laroche (1999) suggest a phase vocoder where peaks are identified in each spectrum. When modifying the spectra, the peaks are treated as units, which ensures that the relative phases and amplitudes in their respective regions of influence are preserved. By this approach, modifications can easily be applied to any set of selected peaks, which gives a much wider range of possible applications.

Using the concepts introduced above, the aim is to investigate the possibilities to change a given note in a polyphonic recording, primarily of a single instrument, and piano music is chosen to be the main test data. The (somewhat fuzzy) measure used will be the perceived audio quality – i.e. how "natural" the music sounds after the modification – in relation to the computational cost. Three aspects will receive the main focus in the tests:

- Roughly speaking, the shorter the sample sequence used for the DFT, the higher the lowest frequency one can distinguish, and the fewer the frequencies that can be resolved. This dichotomy is well understood, but will be present in much of the discussions throughout this paper. The question that inevitably

arises is how to find the optimal frequency/time-resolution trade-of, and how (if at all) it depends on the characteristics of the sound being processed.

- When modifying the spectral content, the result can be significantly improved by the use of different interpolation methods. Some common interpolation techniques will be discussed, and those that are readily implemented will be tested.
- Due to the nature of the DFT, the spectrum obtained from a DFT of a time-limited signal will contain so called *spectral leakage*, which can be reduced by the use of *window functions*. The choice of window function can vary for different purposes, and much has been written on the subject. Here, the suitability of a few well known window functions for the current application will be tested.

Finally, the result of these tests will be put in relation to each other in a discussion about suitable applications, sensible improvements and questions for further investigation.

1.2 Some Key Concepts

In the following review and discussions, some concepts from music theory and digital signal processing are assumed to be familiar. Here follows an brief presentation of the most fundamental concepts.

1.2.1 Mathematics and Digital Signal Processing

There are many ways to refer to a sequence of numbers, depending on the context. Here, the term *signal* will be used to refer to a discrete time signal that act as input or output for some transformation or filtering process. The signal may be obtained by sampling a continuous time signal, or generated in any other way.

In digital signal processing, the term *filter* generally refers to a sequence of numbers that is used to modify a signal, by point-wise multiplication or convolution. Here, the term *function* will be used similarly, referring to a discrete function used for any purpose – including multiplication and convolution with another function.

The Fourier transform (and its inverse) will be used to transfer a signal from the *time domain* to the *frequency domain* (and back), each domain being the reciprocal to the other. When talking about the frequency domain representation of some signal, the discrete time samples $x[n]$ of the time domain are replaced by *frequency coefficients,* $X[k]$, said to reside in *frequency bin* k. Here, the bin index k is a measure of the *time-normalized frequency*, measured in cycles per period, where the period N is the number of samples subjected to the transform – the *analysis period*. The ratio k/N is the *normalized frequency* (ω), measured in cycles per sample, and $2\pi\omega$ is the *normalized angular frequency*, measured in radians per sample. The true frequency f in Hz is given by $f = \omega f_s$, where f_s is the *sampling frequency* in Hz.

1.2.2 Notes and Psychoacoustics

Since most musical instruments generate sound based on standing waves, one usually talks about the spectral content of a note in terms of a *fundamental frequency* and a number of *partials*. The fundamental frequency f_0 is the lowest frequency present in the note, and the k:th partial has a frequency of about $(k+1)f_0$. The fundamental frequency of a note is closely related to our subjective perception of the *pitch* of that note (i.e. how *high* or *low* it is perceived), whereas the partials mainly affect the *character* – or *timbre* – of the sound.

The relative pitch of notes is often referred to in terms of some musical *scale*. In European cultures, the twelve-note chromatic scale is most commonly used, which divides each *octave* – a doubling of frequency – into twelve steps, each step corresponding to multiplying the frequency by the twelfth root of two. This arrangement is used because our perception of pitch is logarithmic – i.e. we perceive the "distance" from 220 to 440 Hz to be equal to that from 440 to 880 Hz. The human ear can detect sound in the range from about 20 Hz to 20 kHz, which corresponds to approximately ten octaves.

Since human ears can detect pressure variations in an enormous range, from about 2×10^{-10} atm to more than 1 atm, a logarithmic scale is again used for sound intensity. Therefore, to get a picture of how a spectrum will be perceived as sound, we shall have to look at the logarithm of the magnitudes.

Chapter 2

Background

Modifying individual notes in a polyphonic recording is theoretically nothing new, though commercial applications have until recently been unavailable. Dolson and Laroche (1999) published a paper in which they suggest a novel implementation of the Phase Vocoder, which appears to be particularly suitable for musical modifications. Here follows a "bottom up" presentation of the standard phase vocoder. This chapter is meant to provide the theoretical framework for the discussions about implementation choices, tests and results in the rest of this essay.

2.1 Fourier Transform

In his book *Théorie analytique de la chaleur* (1822), Joseph Fourier claimed that an arbitrary function f, defined on the interval $[0, l]$, could be written as

$$f(x) = \sum_{k=0}^{\infty} a_k \sin \frac{k\pi}{l} x.$$

Seven years later, in 1829, Peter Gustav Lejeune Dirichlet gave the first correct proof of this claim. This is the foundation of what is today called *Fourier analysis*, and which has been greatly elaborated during the last two centuries (Bennewitz, 2007).

The *Fourier transform* $\hat{f}$ of a function f is defined by

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) e^{-ix\xi} d\xi, \tag{2.1}$$

and its inverse is given by

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{f}(\xi) e^{ix\xi} d\xi. \tag{2.2}$$

Recalling Euler's formula, $e^{i\theta} = \cos\theta + i\sin\theta$, we see that $e^{-ix\xi}$ in Eq. 2.1 can be seen as a periodic function of x with frequency ξ. This means that the complex value $\hat{f}(\xi)$ is the continuous sum over all x of $f(x)$ weighted by a complex periodic function with frequency ξ.

This provides some motivation for the interpretation that $|\hat{f}(\xi)|$ indicates the strength of the periodic component of $f(x)$ with frequency ξ, and that $\arg \hat{f}(\xi)$ gives the phase of this component. Furthermore, Eq. 2.2 shows that f can be recovered from $\hat{f}$.

One interesting point is that if f is real, then $\hat{f}$ will exhibit conjugate symmetry around $\xi = 0$, since

$$\hat{f}(-\xi) = \int_{-\infty}^{\infty} f(x) e^{ix\xi} d\xi = \int_{-\infty}^{\infty} f(x) \overline{e^{-ix\xi}} d\xi = \overline{\hat{f}(\xi)}. \tag{2.3}$$

This is useful to keep in mind, as most of the signals occurring in practical applications are real.

Even if Eqs. 2.1 and 2.2 are very useful for the purpose of analysis, when the transform has to be computed, it is not feasible to work with continuous functions. For such applications, any continuous signal has to be sampled, and we will have to transform a discrete function instead.

2.1.1 The Discrete-Time Fourier Transform (DTFT)

The *discrete-time Fourier transform* (DTFT) of a sequence x is defined by

$$X(\omega) = \sum_{n=-\infty}^{\infty} x[n]e^{-i\omega n}. \tag{2.4}$$

This still gives a spectrum that is a continuous function of the normalized angular frequency ω, but with one important difference: this is a periodic function with period 2π, since

$$e^{-i(\omega+k2\pi)n} = e^{-i\omega n}e^{-ik2\pi n} = e^{-i\omega n}. \tag{2.5}$$

This is an effect of sampling the signal, and the phenomenon is known as *aliasing*. It corresponds to the fact that two sinusoids of angular frequency Ω and $\Omega + 2\pi/T$ sampled every T seconds will have the same value at every sample point; the latter simply completes one extra cycle in T seconds.

Using the DTFT, we can get a spectrum of a discrete function x. However, even if x is has finite length, $X(\omega)$ is still a continuous function, and we will have to sample this spectrum as well.

2.1.2 The Discrete Fourier Transform

Given N sample values $x[n]$, the *discrete Fourier transform* is defined as

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-i2\pi\frac{kn}{N}} = \sum_{n=0}^{N-1} x[n]W^{-kn},$$

where

$$W \equiv e^{i2\pi/N}.$$

The phasor W^{-kn} can be viewed as a periodic function of n with period N/k. Hence, $X[k]$ is a periodic function of k with period N. This means that no more than N different coefficients can be obtained, and this corresponds to sampling the DTFT at the points $\omega = 2\pi k/N,\ 0 \leq k < N$.

The coefficients $X[k]$ form a discrete frequency spectrum of $x[n]$, and k is usually referred to as the *frequency bin number*. Under the assumptions that $x[n]$ is periodic with period N and that the frequencies of its constituent sinusoids do not exceed the frequency represented by $k = N/2$, it can be shown that $|X[k]|$ gives the amplitude and $arg(X[k])$ the phase of the sinusoidal component with time-normalized frequency k (Elliot and Rao, 1982). This corresponds to $f_s k/N$ Hz, where f_s is the sampling frequency in Hz.

Should the first assumption be false, it means that some constituent sinusoid has a frequency that lies between the discrete frequencies represented by k, giving rise to so called *spectral leakage*. Should the second assumption be false, it means that any frequency above $k = N/2$ can be present, giving rise to *aliasing*. Whereas spectral leakage is generally not possible to avoid, aliasing can often be disregarded since it is common practice to low pass filter the signal at the sampling stage to avoid artifacts (Smith, 2009).

When the sample sequence is real then the spectrum will be conjugate symmetric at the origin. Now, since the spectrum has period N, this will also be the case around $k = N$, and hence the spectrum has to exhibit conjugate symmetry around $k = N/2$ as well.

The properties of $X[k]$ mentioned above point to a fundamental limitation in the use of the DFT: when transforming a real signal of length N, one can obtain only $N/2$ useful frequency coefficients, equally spaced between 0 and $f_s/2$ Hz. This means that the longer the sample sequence being analyzed is, the better the frequency resolution in the spectrum will be.

2.1.3 The Inverse Discrete Fourier Transform

To transform the signal back to the time domain, we need an inversion formula, which can be derived by the following relation:

$$\sum_{k=0}^{N-1} X[k]W^{nk} = \sum_{j,k=0}^{N-1} x[j]W^{(n-j)k} = \sum_{j=0}^{N-1} x[j]\{\sum_{k=0}^{N-1} W^{(n-j)k}\}.$$

Looking at the last inner sum, it is clear that $W^k = 1$, when k is an integer multiple of N. For $n = j$, every term in the sum equals 1, so the sum is equal to N. When $n \neq j$, it becomes a geometrical sum with quotient $W^{(n-j)}$, which is different from 1 since $n - j$ never can be an integer multiple of N. Using the formula for geometric sum, the last inner sum becomes

$$\sum_{k=0}^{N-1} W^{(n-j)k} = \frac{W^{(n-j)N} - 1}{W^{(n-j)} - 1} = 0.$$

This means that the entire expression boils down to $Nx[n]$, and the *inverse discrete Fourier transform* (IDFT) is given by

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]W^{nk} = \frac{1}{N}\sum_{k=0}^{N-1} X[k]e^{i2\pi\frac{nk}{N}}.$$

2.2 Spectral Leakage and Window Functions

When using Fourier transform to obtain a spectrum of some finite length signal, usually the resulting spectrum will contain some peaks corresponding to frequencies not actually present in the signal. This phenomenon is often called *spectral leakage*, and it can be dealt with in a number of ways.

2.2.1 The Cause of Spectral Leakage

Each coefficient $X[k]$, $k < N/2$, correspond to a frequency that completes k cycles in N samples, and the normalized frequency $\omega = k/N$ is measured in cycles per sample. If a signal of length N is not periodic with a period that divides N, then at least one constituent sinusoid does not complete an integer number of cycles in N samples. The apparent discontinuity of such a sinusoid, where the signal is assumed to repeat itself, will result in *spectral leakage* in the DFT spectrum.

Spectral leakage can be considered to be an effect of having only a finite sample sequence, since this challenges the periodicity-assumption – as opposed to an infinite sequence, which can be divided by any period. The distribution of spectral leakage, however, can be adjusted through the multiplication of the signal by a suitable *window function*.

2.2.2 Window Functions

A window function can be virtually any function that is zero outside some closed interval. A window (or any signal or filter) that is zero for all negative time-indices is said to be *deterministic*. A common alternative for window functions is to use *zero phase* windows, i.e. to center the window at time zero. Here only deterministic windows will be considered, and for simplicity the nonzero interval is assumed to be $[0, M-1]$, where M is the *window size*.

It should be noted that the window size does not have to be the same as the DFT input size, in which case the signal is said to be *zero-padded*. The simplest window is the rectangular window, which is defined to be one inside the window interval and zero everywhere else, and which simply achieves a truncation of the signal (allowing the same kind of sharp discontinuity at the end points as might be entailed with any finite sequence).

The key to reducing spectral leakage is to reduce the discontinuity between the first and last sample in the sequence. This can easily be achieved by a positive window function that has a central maximum and

approaches zero at the end points. One of the simplest such functions is the *triangular window* (or *Bartlett window*). Many of the most commonly used window functions are "bell shaped" curves, such as the *Hann-* (Figure 2.1) and *Hamming-window*. These functions have the advantage that even their derivative approaches zero at the end points.

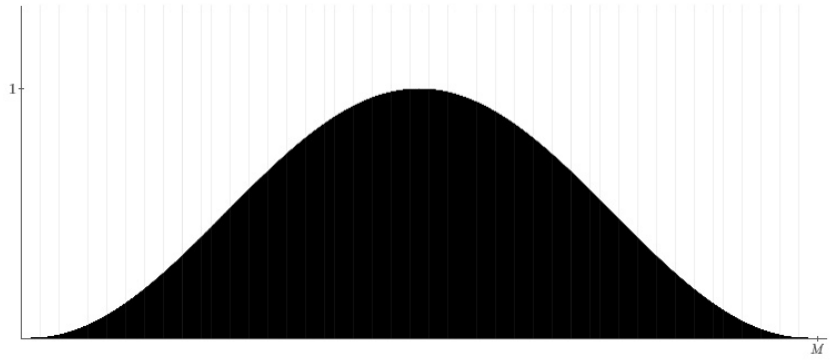


Figure 2.1: A normalized size M deterministic Hann window.

Reducing spectral leakage aims to improve resolution in the spectrum in some sense. However, the ability to resolve two sinusoids of comparable strength that have nearly the same frequency usually comes at the price of not being able to detect a sinusoid of lesser strength, even if its frequency should be at safe distance from its neighbors. This is known as *high resolution* and *low dynamic range*.

A single sinusoid with normalized frequency ω will give rise to spectral leakage that resembles the Fourier transform of the window function, sometimes called the *window kernel*, centered at ωN (by the *Convolution theorem*). The amount of leakage (and resemblance) is determined by the distance to the nearest frequency bin k, the worst case being when ω is located half way between two bins. All the window functions mentioned above (and every other commonly used window) have a Fourier transform that consists of a central main lobe along with smaller side lobes sloping off to the sides. The choice between resolution and dynamic range often comes to a choice between a narrow main lobe and steep slopes in the window kernel.

Actually, most of the preferred windows in spectral analysis have a wider main lobe than the rectangular window and, consequently, lower frequency resolution. This suggests that the main problem when analyzing DFT spectra is that weak sinusoids are lost in the leakage. (If this is as true for musical analysis as for other purposes might be a point to investigate.)

2.2.3 Frequency-Domain Sampling Digression

To see how the window function is related to the spectral leakage, we can look at some properties of the *convolution* operation and the discrete-time Fourier transform.

The discrete convolution of two sequences x and y is a new sequence a which is defined by

$$a[m] = (x * y)[m] = \sum_{n=-\infty}^{\infty} x[n]y[m-n]$$

where the star is the convolution operator, not to be confused with multiplication (Elliot and Rao, 1982). Convolution of continuous functions is similarly defined using integration. It can easily be verified that convolution in the time domain corresponds to point-wise multiplication in the frequency domain, and vice versa, i.e. if $X = DTFT(x)$ and $Y = DTFT(y)$, then

$$DTFT(x * y)(\omega) = X(\omega)Y(\omega)$$

and conversely

$$DTFT(xy)(\omega) = (X(\omega) * Y(\omega)).$$

The N-point DFT of a sequence x can be seen as sampling the spectrum which is given by

$$DTFT(xw)(\omega) = (X(\omega) * W(\omega))$$

at the points $\omega = 2\pi k/N$, where w is a rectangular window of length $M = N$ and $W = DTFT(w)$.

If x is a sampled sinusoid with normalized frequency f cycles per sample, then X will contain a single peak at $\omega = 2\pi f$, which means that the spectrum $DTFT(xw)$ can be described as $W = DTFT(w)$ centered at this point.

The magnitude of the $DTFT$ of a rectangular window w centered at 0 with length N is

$$|DTFT_\omega(w)| = |W(\omega)| = |\frac{sin(N\omega)}{\omega}|,$$

which is clearly 0 at the points $\omega = 2\pi k/N$, except for $k = 0$ when it is 1. The convolution of $W(\omega)$ with $X(\omega)$ will result in the same function shifted to be centered at $\omega = 2\pi f$. If $f = k/N$ for some integer k, it will still be 0 at all the sample points except one, $k = fN$. However, if $k/N < f < (k+1)/N$ for some integer k, all the zeroes will end up between the sample points, the result being spectral leakage. Since any length N sample sequence can be considered multiplied by a length N rectangular window, this argument shows that spectral leakage occurs only if the analyzed signal contains sinusoids with a period not dividing N.

2.3 Spectrograms and the Short-Time Fourier Transform

As mentioned before, the DFT of some discrete signal $x[n]$ will tell us the amplitude and phase of the sinusoids of discrete frequencies that would sum up to the entire signal. To obtain a time varying spectrum, or *spectrogram*, one has to apply the DFT to many shorter time-segments of the signal.

2.3.1 The Short-Time Fourier Transform (STFT)

To obtain a spectrogram of a signal, many shorter, evenly spaced segments, called *analysis frames*, are transformed one at a time. This segmentation can be viewed as applying a window function $w[n]$ to the original signal at some regular interval. One could use different windows for different frames, but this is rarely practical so we assume the window to be the same for all frames.

The length M of the interval outside of which the window function is zero is called the *window size*, and the number of samples R that are skipped between two adjacent frames is called the *hop size*. In the following it is assumed that the interval in which the window is nonzero is $[0, M-1]$. The short-time signal $x_m[n]$ of frame m can then be defined as

$$x_m[n] = x[n]w_m[n] = x[n]w[n - mR].$$

The short-time Fourier transform is usually defined using the DTFT. Since this is rarely practically useful, the discrete (or *frequency sampled*) STFT of frame m can instead be defined by

$$X_m[k] = \sum_{n=-\infty}^{\infty} x_m[n]e^{-i2\pi\frac{nk}{N}} = \sum_{n=-\infty}^{\infty} x[n]w[n - mR]e^{-i2\pi\frac{nk}{N}} = \sum_{n=m}^{m+M-1} x[n]w[n - mR]e^{-i2\pi\frac{nk}{N}},$$

since the window is zero outside $[0, M-1]$. The coefficients $X_m[k]$ tell us the amplitude and phase of the sinusoidal component with normalized frequency k/N in the m:th frame.

Due to the nature of the DFT, one would like to choose a large value for M since this gives a better frequency resolution. On the other hand, R should preferably be small since this gives better time resolution (most importantly, the hop size should not exceed the window size, or some samples would not be included in any analysis frame).

Not surprisingly, both time and frequency resolution comes at a greater computational cost; half the hop size result in twice the number of DFTs, and twice the window size implies twice the DFT size, resulting in more than twice the number of computations per DFT.

The ratio between R and M determines the amount of overlap between consecutive frames, and the ratio $(M-R)/M$ is called *the overlap ratio*. This also has some implications to the spectrum obtained, and it is important for the resynthesis phase.

2.3.2 Inverse STFT and Overlap-Add

A window function w that satisfies

$$\sum_{m\in\mathbb{Z}} w(n-mR) = 1, \forall n \in \mathbb{Z}$$

is said to satisfy the *constant overlap-add* (COLA) condition for hop-size R. This is a very useful property when dealing with the STFT; it ensures that the original signal can be reconstructed from the individual frames by a simple summation, as can be seen by

$$x[n] = \sum_{m=-\infty}^{\infty} x_m[n] = \sum_{m=-\infty}^{\infty} x[n]w[n-mR] = x[n] \sum_{m=-\infty}^{\infty} w[n-mR].$$

In other words, the *inverse STFT* (ISTFT) is performed by simply running the IDFT frame by frame and summing up the result.

If the window size is not specified, it is better to consider the overlap ratios for wich the window type is COLA. For instance the Hann windows is COLA for 50% overlap, as can be seen from Figure 2.2.

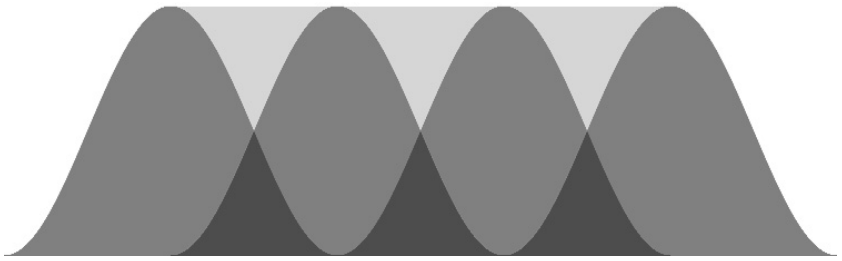


Figure 2.2: A Hann window giving constant overlap-add for 50% overlap.

2.3.3 Synthesis windows

When the spectral content has been modified before the reconstructive inverse transform is applied, there is a great risk of audible discontinuities appearing at the boundaries between frames. Such artifacts can be greatly reduced by applying another window function after the inverse DFT, before the final overlap-add operation.

Since the purpose is the same as with analysis windows (i.e. suppressing discontinuities at the frame end points), similar windows can be used. The simplest way to find windows that provide COLA for this setting is to reformulate the condition above:

$$\sum_{m\in\mathbb{Z}} w^2(n-mR) = 1, \forall n \in \mathbb{Z}.$$

Any window that satisfies this condition can be used both as analysis and synthesis window. Many of the most common, bell-shaped windows are all positive, so they can easily be adjusted to this condition simply by taking their square root.

2.3.4 Zero-padding

Many DFT-algorithms only work for certain values of N (typically powers of two). One common solution to this is to pad the end of the signal with zeros to reach the next acceptable sequence length. This means that no new data is used as input to the DFT, yet the number of frequency coefficients obtained has increased. There is a simple explanation for this: zero padding in one domain corresponds to optimal interpolation in the reciprocal domain (Smith, 2009).

Zero-padding becomes even more interesting in the context of the STFT, since it makes it possible to increase the DFT size to obtain optimal interpolation, while keeping the window size constant to preserve the desired overlap ratio.

2.4 The Phase Vocoder

The *phase vocoder* is a powerful technique for time- and pitch-scale modifications of audio. It has been developed from voice coding methods originally aimed at compressing data, and the word "vocoder" is actually a contraction of the term "voice coder". It was introduced in 1966 by Flanagan and Golden, and in the last 20 years it has received great attention for its wide applicability in spectral analysis.

2.4.1 The Standard Technique

At its core, the phase vocoder relies on the STFT in the transitions from signal to spectrum and back again. To modify the spectrum with a minimum of undesired artifacts in the re-synthesis, the phase propagation between analysis frames has to be considered. Indeed, more accurate information about the constituents true frequencies can also be obtained from this phase propagation.

Phase unwrapping

A sinusoid with frequency $f_s k/N$ completes k/N cycles per sample, which means that its phase propagation between two adjacent frames R samples apart will be $2\pi kR/N$. If $X_m[k]$ (the k:th coefficient in the m:th analysis frame of the STFT) draws its main contribution from a constituent sinusoid whose frequency is really $f_s k/N$, then the phase of $X_{m+1}[k]$ is indeed given by

$$arg(X_{m+1}[k]) = [arg(X_m[k]) + 2\pi k\frac{R}{N}]_{2\pi}$$

where the subscript by the closing bracket indicates that the number should be *wrapped* in the interval [0,2π). Any deviation from this *expected phase propagation* will be proportional to the deviation of the true underlying frequency from the frequency bin (Dolson, 1986).

The calculation of the expected phase propagation and any deviation from it is often referred to as *phase unwrapping*. This is because calculating the expected phase propagation can be seen as looking at how the phase would develop with time if it were not wrapped in the interval [0,2π).

Time-scaling

To alter the duration of a signal while preserving its spectral content, one can simply change the hop size R before the ISTFT. If R_1 and R_2 are the hop sizes for the STFT and ISTFT, respectively, then setting $R_2 = 2R_1$ will effectively double the duration of the signal. However, the phase propagation between $X_m[k]$ and $X_{m+1}[k]$ should also be doubled to preserve information about the corresponding frequency (Dolson, 1986).

Pitch-shifting

To modify the frequency content of the spectrum, similar considerations are necessary. To move $X_m[k]$ and $X_{m+1}[k]$ from bin k to bin αk in their respective frame corresponds to a shift by $(\alpha - 1)k$ bins (α can be called the *scale factor*). Then the corresponding sinusoid completes α as many cycles between consecutive frames as

it did before the shift, and the phase of $X_{m+1}[k]$ has to be adjusted accordingly. This is conveniently done by multiplying $X_{m+1}[k]$ by $e^{i(\alpha-1)k\frac{R}{N}}$, where like before R is the hop size and N is the size of the DFT (Dolson and Laroche, 1999).

2.4.2 The Peak-Based Phase Vocoder

The new technique described by Dolson and Laroche (1999), like the standard phase vocoder, uses the STFT to obtain a spectrogram of the signal. The main idea is then to identify peaks in each time frame, each having some *region of influence* in the frequency domain. Now, changing the frequency of some peak can be easily done by applying the same shift and phase adjustment to each bin in the peak's region of influence.

The underlying idea has to do with with the fact that the human ear is sensitive to peaks in the spectrum, and a linear shift of an entire peak-region will preserve the relative phases and amplitudes for the bins in that region, thus better preserving the frequency-variations exhibited by the sinusoid thought to be represented by this peak.

In its simplest form only integer shifts are allowed, and no more considerations are necessary. However, for fractional shifts some kind of interpolation is needed. Dolson and Laroche (1999) suggests linear interpolation using 75% overlap, and point out that this will introduce sidebands about -51 dB below the peak magnitude in the worst case (a half-bin shift). This will be inaudible if the frame rate is the order of tens of Hz, and can be expected to produce satisfactory result for a great range of the spectral modifications considered here.

Chapter 3

Implementation

In this chapter, the most important implementation choices are described along with their motivation. The outline follows that of the implementation process, which can be divided into four major phases. The first three correspond roughly to the DFT, STFT and phase vocoder. The final phase is more general, and covers any additional functionality to facilitate the desired modifications and testing.

3.1 Computing the Discrete Fourier Transform

The most fundamental building block of the application is the computation of the DFT. A fast Fourier transform algorithm is selected to do this efficiently.

3.1.1 The Fast Fourier Transform (FFT)

Computing the discrete Fourier transform of size N explicitly from the definition, one has to find N values, each being the sum of N terms, so $O(N^2)$ computations are required. However, there is a family of well known algorithms, known as *fast Fourier transforms* (FFT), that can accomplish this in $O(N \log N)$.

The most famous FFT is the Cooley-Tukey algorithm, which recursively divides a DFT of composite length $N = N_1 N_2$ into smaller DFT:s of length N_1 or N_2. A particularly simple case of this algorithm, the radix-2 decimation-in-time FFT, can be applied when N is a power of two. Usually the length of the data sequence can be extended arbitrarily by appending zeros, so this is generally no significant restriction. (Elliot and Rao, 1982)

3.1.2 Radix-2 Decimation-in-Time FFT

The radix-2 decimation-in-time (DIT) FFT algorithm was first published in 1965 by Cooley and Tukey. In a DIT FFT, the divide-step of the algorithm is applied to the input – i.e. in the time-domain instead of the frequency domain. In the case of radix-2, the input sequence of length N is split into two interleaved sequences, each of length $N/2$. This step is then repeated until every subsequence has length 2, which is why N is required to be a power of two. (Hayes, 1999)

This algorithm is chosen to perform the DFT computation, and any data sequence of length other than a power of two is simply zero padded.

How it works

The basic principle of this algorithm relies on what is known as *the Danielson-Lanczos lemma*, which was proved by Danielson and Lanczos in 1942. It states that a DFT of length N can be reformulated as the sum of two DFT:s of length $N/2$, one consisting of the odd-numbered points, the other of the even-numbered points. The proof is as follows:

$$X_k = \sum_{n=0}^{N-1} x[n] e^{-i2\pi \frac{nk}{N}} = \sum_{n=0}^{N/2-1} x[2n] e^{-i2\pi \frac{k(2n)}{N}} + \sum_{n=0}^{N/2-1} x[2n+1] e^{-i2\pi \frac{k(2n+1)}{N}}$$
$$= \sum_{n=0}^{N/2-1} x[2n] e^{-i2\pi \frac{kn}{N/2}} + W^k \sum_{n=0}^{N/2-1} x[2n+1] e^{-i2\pi \frac{kn}{N/2}} = X_k^{even} + W^k X_k^{odd}$$

where $W \equiv e^{i2\pi/N}$, and X^{even} and X^{odd} is the size $N/2$ DFT of the even and odd numbered points of $x[n]$, respectively (Flannery et al., 2007).

To do this recursively one would have to allocate memory for the subsequences in each recursive call. Instead one can do the calculations in place by changing the indices. This can be done by reversing the bit-order of the original indices. This can be seen as taking the least significant bit (determining if the number is odd or even) and making it the most significant bit. Ordering the numbers now, one will get the numbers that were even first, followed by the odd numbers, in order. This is repeated by taking the bit that was the second least important, making it second most important etc. Arranging the data according to these new indices, and by exploiting the Danielson-Lanzcos lemma, one can make the computations in place.

3.2 Short-Time Fourier Transform and Window Functions

Moving up one level in the application, we apply the FFT to windowed segments of the analyzed signal, which brings us to the STFT and the implementation of the window functions.

3.2.1 Choice of Window Function

The choice of window function is not a crucial point in the implementation process. As they are easily implemented and easily exchanged, yet a nuisance to analyze, the only reasonable decision at this point is to leave it open. Though much can be said about the different window properties, the suitability for a given task still partly depends on characteristics of an input signal that may not be known in advance. Still, some windows seem more interesting to investigate than others, allowing us to narrow the choice down to a few strong candidates.

Since the application involves resynthesis of the analyzed signal, the COLA condition has to be considered, and since a synthesis window is also needed, the window should be all positive so that its square root can be used.

As mentioned before, the rectangular window provides good frequency resolution but poor dynamic range. It is interesting, partly because of its simplicity, but also because of the high resolution and the fact that it is COLA for many overlap ratios (and even equal to its own square root). Another simple window of interest, which is also COLA for many settings, is the triangular window. This has a main lobe that is twice as wide as that of the rectangular window, but the amplitude of the first side-lobes are twice as far down from the main lobe.

The most common choices for spectral analysis are *Hamming* and *Hann* windows, both belonging to the *raised cosine* window family. They provide fairly good frequency resolution along with a high dynamic range. The amplitude of the side-lobes decrease faster with the Hann window, whereas the highest side-lobes become smaller with the Hamming window. Another famous window type with a shape similar to the Hann window, but with slightly better resolution and less dynamic range, is the Gaussian windows. Though these windows lack the COLA property except for the case of a sliding window (i.e. when the hop size is equal to one, in which case any window is COLA), one version is implemented here for comparison.

Finally, a low-resolution window, *Blackman-Harris*, which can be made to satisfy the COLA condition for 75% overlap, is also implemented.

Table 3.1 lists the windows that are implemented, along with the definitions used, where as before, M is the window size, and it is assumed that $w[n] = 0$ for $n \geq M$.

3.2.2 Implementing the STFT

The STFT can be implemented more or less straight from the definition. Only how to treat the first and last frames requires some extra consideration. To get the COLA property for the first and last frame, we will have

Table 3.1: Window definitions

Window	Definition
Rectangular	$w[n] = 1$
Hann	$w[n] = 0.5\ \left(1 - \cos\left(\frac{2\pi n}{M-1}\right)\right)$
Hamming	$w[n] = 0.54 - 0.46\ \cos\left(\frac{2\pi n}{M-1}\right)$
Triangular	$w[n] = \frac{2}{M} \cdot \left(\frac{M}{2} - \left\lvert n - \frac{M-1}{2}\right\rvert\right)$
Blackman-Harris	$w[n] = 0.35875 - 0.48829\cos\left(\frac{2\pi n}{M-1}\right) + 0.14128\cos\left(\frac{4\pi n}{M-1}\right) - 0.01168\cos\left(\frac{6\pi n}{M-1}\right)$
Gaussian	$w[n] = e^{-\frac{1}{2}\left(\frac{n-(M-1)/2}{0.4(M-1)/2}\right)^2}$

to apply a size M window less than M sample points from the beginning and end of the sequence. This can be done by zero padding both ends of the sequence.

Since this implementation relies on a radix-2 FFT, we decided at an early stage to apply zero padding at the end of any signal that is not already of appropriate length. Therefore, a simple solution was to apply the STFT in a cyclic manner (i.e. letting the last frame extend into the beginning of the sequence). When the zero padding extends at least one window size and the hop size divides the sequence length, this effectively corresponds to zero padding both ends. Should these conditions not be satisfied, some fluctuation in amplitude might occur in the beginning and end of the resynthesized signal.

To make it easier to change window function and the amount of zero padding in the STFT, the different windows are implemented as different instance classes of a common interface; a window takes the window size as parameter to the constructor and has the window function itself as its only function, taking a reference to the value to be scaled along with its index in the frame as parameters. If the index exceeds the window size, it is set to zero. In this way, for any given data sequence, the STFT needs only a pointer to some window function, the DFT size and the hop size as parameters, implicitly giving the type of window function, window size and amount of overlap and zero padding to be used.

3.3 Phase Vocoder Issues

At the highest level of this implementation, the phase vocoder can be seen as the tool to modify spectrograms produced by the STFT.

Before any modification can take place, the spectrogram has to be read and analyzed. Here, it is exploited that the original signal is real, and only the first $N/2+1$ coefficients in each frame are read, discarding the mirrored complex conjugate. The next step is to detect the peaks.

3.3.1 Peak Detection

Defining the peaks can be done in a number of ways, and very simple peak detection has been found to work well in practice. Dolson and Laroche (1999) suggests that a bin is a peak if its magnitude is greater than its two neighbors on each side. This will be used in the current implementation, along with two similar definitions, where a peak is greater than one or three neighbors on each side, respectively.

There are two main aspects to consider in the evaluation of these definitions: the ability to resolve peaks of similar frequency, which can be of great importance among the low frequency bins, and, on the other hand, the number of peaks detected – ideally, only the peaks in the continuous spectrum are identified, and side lobe peaks resulting from the nonperiodicity the signal are discriminated.

3.3.2 Interpolation

To shift a peak at p to a new location αp, i.e. a shift by $(\alpha - 1)p$ bins, we can pick interpolated values around p and put them in the corresponding bins around αp. One simple way to do this, which is implemented here,

is by picking the value for the last bin before αp from the line between the bins at $p-1$ and p. However, this will result in a lower peak maximum, and the real location of the peak is not considered.

A better way would be to first approximate the true location and maximum value of the peak by fitting a parabola, before using linear interpolation (Smith and Serra, 1987). This solution is not fully implemented here, but the peak location can be found by this technique. An improved approximation of the peak location can be useful, even if no interpolation is used, since the phase can be adjusted to account for a shift by a fractional number $(\alpha-1)p$.

Here, three versions of peak shifting with integer amplitude shift and linear phase adjustment are implemented. The simplest form of linear phase adjustment is to apply a phase shift corresponding to a shift from bin p to the location $(\alpha-1)p$. Another way is to calculate the phase shift from the location p_i, obtained by fitting a parabola, to correspond to a shift from p_i to $(\alpha-1)p_i$. Finally, phase unwrapping can be used to find the true peak locations from which the phase adjustment is calculated.

3.3.3 Peak Shifting

Spectral modifications are performed using a result buffer, which is initially zeroed. Uniform frequency scaling is performed by shifting all the peaks to their new location in the result buffer, which is then set to be the main data and a new result buffer is initialized.

For single peak shifts, the coordinates of the peak are passed along with the scale factor, and this peak is moved to its new location in the result buffer. When all the desired shifts are performed, the remaining unmodified data is simply copied to the result and the buffer is set to be the main data.

According to Dolson and Laroche (1999), the phase adjustment has to be incremented from frame to frame when shifting the peaks. This was found to be easier said than done. To shift a selected peak, this peak has to be shifted from start to end to ensure phase coherence. This brings up the question of how to define peaks in the time-direction (only the frequency-direction is discussed in the article).

For uniform shifts, this is treated by always incrementing the adjustment applied to the same bin in the last frame, regardless of whether that bin belonged to the same peak. For single peak shifts, two alternatives are implemented. In one case, the selected peak is considered to extend backward and forward as long as that bin is a peak, the entire peak is shifted, and the phase adjustment is incremented from the first frame to the last for that peak. In the other case, only the peak in the selected frame is shifted and the increment between frames simply disregarded.

3.3.4 Time Scaling

Time scaling is not directly related to the main purpose of this application, but it is relatively simple to implement and can be of interest for evaluation of uniform frequency scaling. Pitch shifting with the traditional phase vocoder is done by time scaling followed by resampling. Here, pitch shifting is done by shifting all peaks in the spectrum, which can be expected to introduce more artifacts than the standard technique. Resampling, however, is not implemented here, since this requires an entirely different algorithm, and resampling the time-scaled signal has to be done with a different application.

Time scaling is done by changing the hop size between the STFT and ISTFT. To account for the altered duration between frames, the phases of all the coefficients in each frame are adjusted accordingly, which is straightforward to do from the theoretical discussion.

3.4 Additional Functions

The functions presented so far cover the fundamental operations of interest. Some additional functionality is required to tie the application together. The most important functions to make the application useful are presented here.

3.4.1 Displaying Spectrograms

The main feature of the GUI is to display spectrograms from which peaks can be selected by the user.

Spectrograms are displayed by mapping the magnitudes of the coefficient values to a greyscale, and producing a pixel matrix that fits the display window. This is done without any interpolation, i.e. if the spectrogram is too big, some values are skipped, and if it is too small, some adjacent pixels get the same value. As usual, time is displayed horizontally starting to the left and frequencies are displayed vertically with the lowest frequencies at the bottom. Since the maximum amplitude is not known in advance, the values are normalized by mapping the largest value to a white pixel (here, one byte is used for each colour value, so red = green = blue = 255).

Looking at the result from the STFT directly, usually most peaks will be too weak to be visible. The logarithm of the values are usually in a better range. Taking the logarithm of the values in the spectrogram is implemented in a separate function that has to be called explicitly by the user. Actually, the complex logarithm is used, but this is a good enough approximation as only the magnitude is considered.

As most of the interesting peaks are located in relatively low frequencies, the number of frequencies to read from the STFT is determined by a number p that can be set by the user, and only $N/2^p$ frequencies are displayed. As the image size is fixed, this is effectively a way to zoom in at the low frequencies. Figure 3.1 shows a spectrogram of a piece of piano music produced by the application, where one fourth of the frequencies analyzed are displayed.

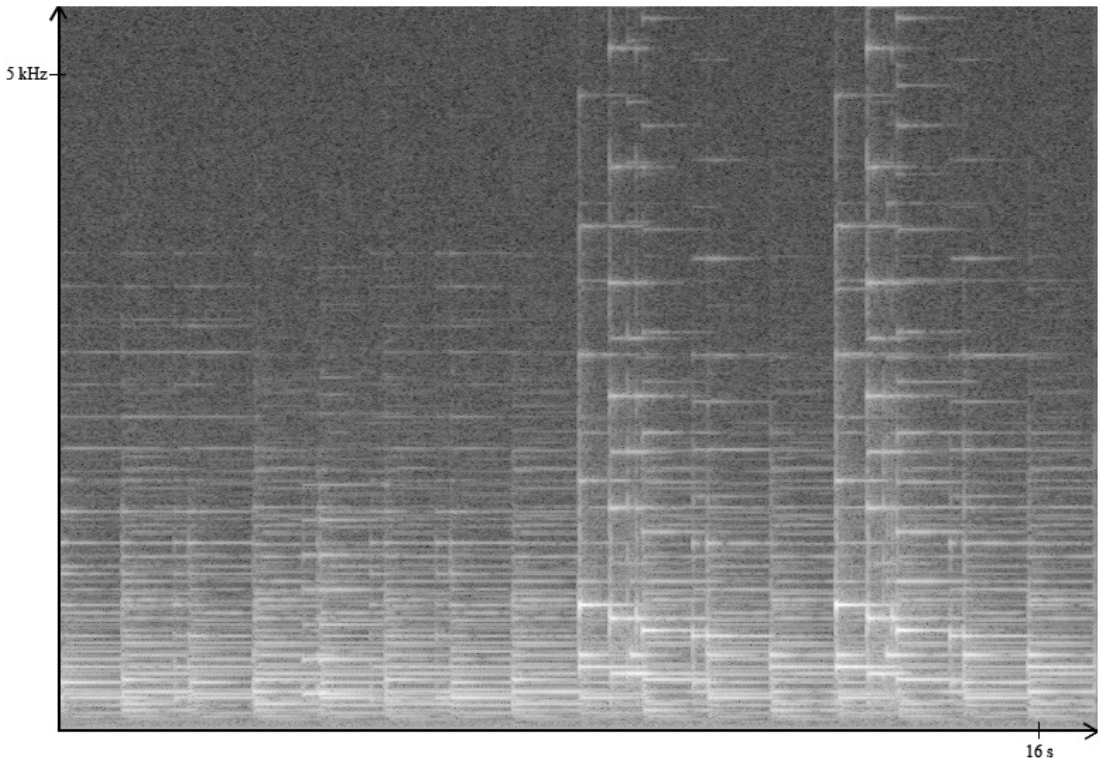


Figure 3.1: A spectrogram corresponding to 16 seconds of *Funeral March* by Chopin, obtained using a 4096 size square root Hann window with 75% overlap, showing the frequencies from 0 to about 5.5 kHz.

3.4.2 Peak Selection

Peaks are selected by selecting a range in the displayed spectrogram, which is done using the mouse pointer. Clicking a mouse button gives one point of a selection rectangle, and the point where the button is released gives the opposite corner. One useful feature with this approach is the ability to easily select all peaks at a given frequency.

Harmonic peak selection

Since the goal is to achieve "musical" harmonic modifications, a natural step would be to implement some kind of grouping of detected peaks into notes according to the model of fundamental frequency and partials. Here, only a very basic detection of octave peaks is implemented: when selecting a peak at frequency f_0, all doubles of the frequency can be checked (i.e. $2f_0$, $4f_0$, $8f_0$, etc.), and if a peak is found at any of these frequencies, it is moved by the same amount.

3.4.3 File Handling

Sound files

To make use of the implementation, we need to read and write sound data according to some standard that can be used for playback by other applications. Here, the Microsoft WAVE PCM sound file format as described by Sapp (2003) is used. This seems to be a suitable choice as it is simple to parse, yet widely used and recognized by most media applications. Full support is implemented for mono sound sampled at 44100 Hz and 16 bits per sample, as this is a standard bit rate for music, and since channels have to be processed separately even if more than one are present.

Image files

As the appearance of the spectrograms is of some interest in itself, support for writing the content of the display to TGA image files is also implemented. This format has a simple representation of pixel values. The specification described by Bourke (1996) is used. Though the result is a black and white image, it is saved as an uncompressed RGB image for simplicity. Figure 3.1 is originally saved as a TGA file by the application.

3.4.4 Timer

To get an idea of the time used for the main processing, a timer function is implemented. Every time some transformation, modification or analysis step is performed, the timer is activated. These steps include the STFT/ISTFT and all of the vocoder operations, but nothing related to displaying the data, such as normalization or taking the logarithm.

3.5 Application Overview and Usage

The most interesting functions implemented are described above. Here follows a description of the general design of the application and how it is intended to be used.

3.5.1 General Design

All internal data is assumed to be complex, and can be either a vector or a matrix. There are four main parts of the program where data can be stored and/or processed. Here, these will be called the controller, the transformer, the vocoder and the GUI.

When a file is parsed, the data is stored in the controller, and from there it can be passed to the transformer for processing or to the GUI for display. Here, the data can also be normalized, and the logarithm can be taken of all values. The data stored in the controller can be saved to files (vector data to a .wav file and matrix data to a .tga file). The settings used for the current transformations and modifications are also stored in the controller, including transform size, window size, hop size, type of window and the number of semitones to shift peaks.

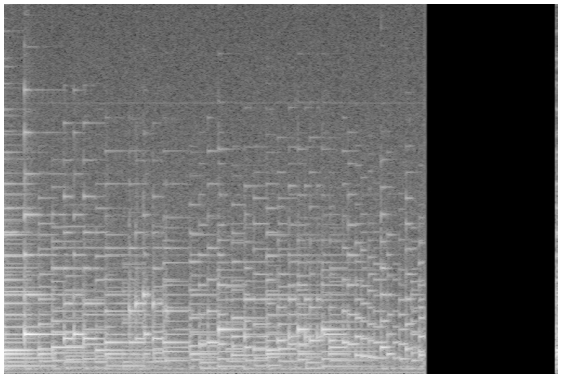
Vector data that is stored in the transformer can be subjected to the STFT or read by the controller. Matrix data that is stored here can be subjected to the ISTFT, passed to the vocoder, or read by the controller.

Once some matrix data is passed to the vocoder, pitch modifications are possible, and should the hop size be adjusted before the data is written back to the transformer, time scaling is obtained.

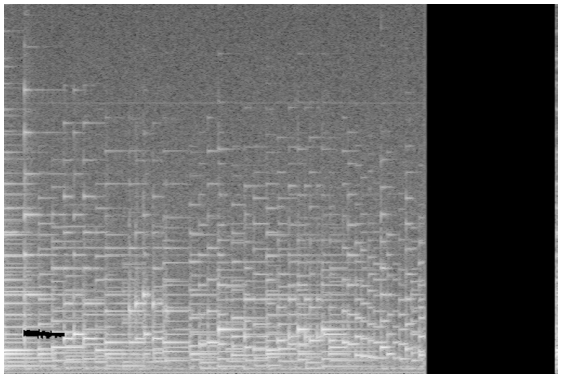
3.5.2 Common Usage Example: Shifting a note

First, the sound file to read from is passed as an argument to the application from the command line. This file is automatically parsed, and the initial sound data is stored in the controller. Next, the user passes the data to the STFT buffer. Before the STFT is applied, the settings can be adjusted. Once a spectrogram is obtained, it can be read back to the controller. Here, it is usually desired to take the logarithm of the values before displaying the result, as only the strongest peaks are visible with linear amplitude (see Figures 3.2a and 3.2b).

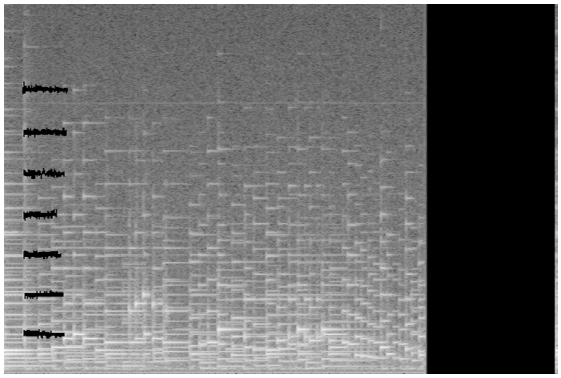
(a) The original spectrogram with linear amplitude. Only some of the strongest peaks are visible.



(b) The original spectrogram with logarithmic amplitude. The black region is a result of zero-padding.

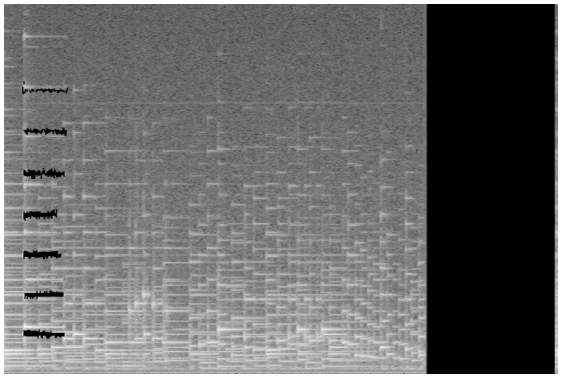


(c) A peak, corresponding to the fundamental frequency of the note to be shifted, has been moved to the result buffer.



(d) The six most prominent partials have been moved to the result buffer.

(e) The result buffer, now containing seven peaks shifted up by 3 semitones.



(f) Here, the result buffer has been added to the rest of the unchanged spectrogram.

Figure 3.2: Spectrograms of a piece of piano music at different stages of a note-shift operation.

Now, to modify the spectrum, the spectrogram has to be read by the vocoder. When this is done, the desired amount of shift can be set, and the peaks to be shifted can be selected from the displayed image. To see the result of a peak shift operation, the vocoder data has to be written back to the transform and read by the controller.

Any shifted peak will leave a black region in the displayed image (see Figures 3.2c and 3.2d). Alternatively, the result buffer of the vocoder can be read, in which case everything not shifted will appear black (Figure 3.2e).

Figure 3.3 is a screenshot showing the appearance of the application after shifting a peak. The order of operations is similar to the one just described.

When the desired peaks are shifted, the unmodified spectral content has to be added to the modified peaks (see Figure 3.2f), which is done by shifting all peaks by zero semitones. Now the result can be written to the transformer, the ISTFT can be applied, the resulting vector read back to the controller, optionally normalized, and saved to a .wav file.

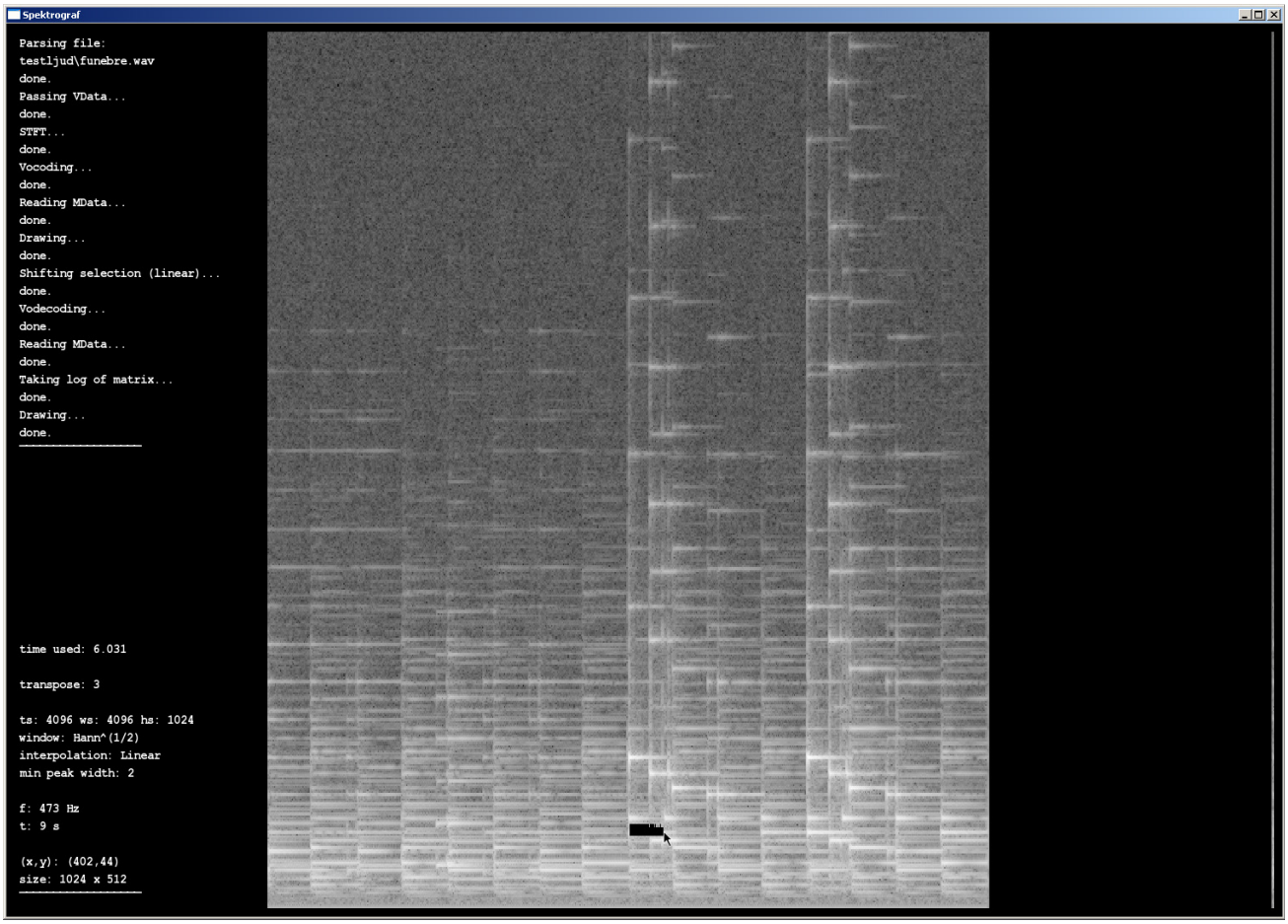


Figure 3.3: A screenshot of the application after some typical operations. The command history can be seen to the left, below which the current settings are displayed and the time used, in seconds, for analysis, modification and resynthesis. At the bottom left the mouse pointer location is indicated – as matrix index, and as frequency and time. The peak in the black region is located at approximately 470 Hz, about 9 seconds from the time origin. It has been shifted by 3 semitones and saved in the result buffer, which is not shown here.

Chapter 4

Tests and Results

The main focus of the tests is to shift all or parts of the spectral content of different kinds of recorded music. The aim was initially to be able to achieve arbitrary spectral modification of single instrument recordings, mainly piano. However, as the application has proven itself useful to a much wider range of music, the result of some other tests will also be presented.

The result is evaluated by how close it appears to an original recording of the desired harmonic content. For larger frequency scale factors, this is an increasingly fuzzy measure, since the pitch shifting itself will make the instrument sound unnatural. However, one can usually tell the effect of pitch shifting from undesired artifacts and noise and effects such as amplitude and frequency modulation.

Any mentioning of noise, artifacts or the like in this presentation refers to what we have perceived when listening to the results, using headphones and speaker systems of high fidelity.

All test data used is sampled at 44100 Hz and 16 bits per sample. The tests are run on a computer with a 2.0 GHz processor, 2.0 GB RAM, running Windows XP.

4.1 Setting the Preferences

Here, the different parameters of interest are covered one by one. Since they all relate to each other, there is no strict partition, and the sections overlap to some extent. Mostly uniform pitch shifts of the entire signal are used for these tests, as single note shifts are affected by the selection of peaks, which is done manually.

As test data, some different pieces of piano music with disparate harmonic complexity have been used – i.e. from few notes played slowly to many notes played fast. Since an acoustic recording is hard to analyze and may give rise to unpredictable effects by some unknown property of its spectrum, some synthetic sounds are used as reference. The reference sounds are a sine wave with a frequency of 220 Hz, and three square waves with fundamental frequencies 10, 100 and 1000 Hz, respectively. [1]

4.1.1 Choice of Window Function

Using a rectangular window introduces far more noise than any of the more sophisticated window functions. Looking at the spectrogram, the peaks obtained with a rectangular window are thin, but the picture is also very bright in general, due to the spectral leakage. The Hamming and Gaussian windows also introduces more noise than the other windows, in accordance with the leakage that can be observed in the corresponding spectrogram.

For the Blackman-Harris window, this kind of noise is not an issue. However, when peaks appear close to each other in the spectrum, frequency distortion often occurs. It appears that the frequency resolution is simply insufficient to resolve the peaks.

This leaves us with the Hann and Triangular windows. These perform very well under most circumstances, and it is actually hard to tell any difference for many settings. We found that the triangular window introduces

[1] The result of spectral modifications to square waves is quite interesting to analyze because the shape of the wave is dependent on a very precise relation between the fundamental frequency f_0 and the partials of the note. A square wave can be approximated by sine waves with frequencies f_0, $3f_0$, $5f_0$, $7f_0$, etc. Even the smallest change to their relative positions in the spectrum will become apparent in the result. Therefore, they provide a sensitive indicator for frequency distortion in the modification.

slightly more noise for some signals when using 75% overlap. For a 50% overlap, however, better results are obtained for a triangular window. Using a Hann window with this setting seems to produce some amplitude modulation that is less audible when using a triangular window.

Constant overlap-add

One thing that came as a pleasant surprise is that the COLA property, though important for mathematical correctness, can be taken quite lightly in practice. For instance, the Gausian window can give reconstruction without audible fluctuations even for 50% overlap.

Synthesis windows

We found the use of a synthesis window to have a major positive impact on the result, as soon as the pitch shift is larger than a few semitones. Fortunately, the appearance of the spectrogram is generally only slightly affected by the use of a square root window instead of the original form, which indicates that we can use the square root version of the functions without any greater impact on the result. One exception to this is the Hamming window; as it has quite low dynamic range, a lot of energy is widely spread in the frequency bins, and the entire spectrogram is much brighter due to leakage when using the original form instead of the square root.

4.1.2 DFT Size, Overlap and Zero-Padding

Piano music proves to be very difficult to pitch shift because of the sudden attack of the notes, and it is difficult to find settings for which both frequency and time resolution is sufficient. Especially when the shift is large, uniform pitch shifts call for both zero padding and large overlap at a high computational cost.

There is generally a very audible difference between using a 50% and a 75% overlap. Using a triangular window and small pitch scale factors, 50% might suffice, but for any larger modifications 75% can be considered a minimum. For overlap ratios larger than 87.5% (i.e. at less than half of the hop size for a 75% overlap), the attack of the notes is usually smeared and reverberation occurs in way similar to when the hop size and window size are too big. In these tests, the best results are obtained for overlap ratios of 75-87.5%.

As for the DFT size, 4096 samples (corresponding to about 93 ms) seems to suffice for most shift operations, though 2048 might suffice when the scale factor is small or the spectrum contains few peaks. This would suggest a hop size of about 512-1024 when no zero padding is used. Again, for small scale factors this might be good enough, but for larger shifts, the attack of the notes will be smoothed considerably.

Zero-padding provides some help to solve dichotomy of frequency versus time resolution. The use of zero-padding could logically be treated as an interpolation method, but as it is set by the parameters discussed here, and since it is computationally more expensive than the other interpolation methods considered in this implementation, it fits into this section as well.

When a good frequency resolution is found for some given operation, but the time resolution is insufficient even for 87.5% overlap, it often helps to reduce the window size along with the hop size, leaving the DFT size unchanged. Conversely, if a good time resolution is found but the spectral resolution is not sufficient, the DFT size can be increased, leaving the window size and hop size unchanged.

Considering what has been found about DFT size and overlap ratios, promising settings for most operations involve a DFT size of 2048-4096 and a hop size about 256-512. It has been found that results of such operations are often slightly improved by using a DFT size of 4096 and a window size of 2048, which gives 100% zero-padding. Using more zero-padding is not thoroughly tested, as this is simply too time consuming compared to the conceivable gain.

4.1.3 Interpolation

Three different approaches are compared here; integer shifts (no interpolation), linear interpolation, and linear phase adjustment (i.e. fractional phase shifts with integer amplitude shifts). The difference between them

becomes less obvious with larger windows. Most tests here are performed with widows in the range 1024-4096.

Integer shifts

Restricting the spectral modifications to integer frequency shifts will demand a large transform- and window size. Using zero padding can improve the result, but it is not computationally efficient, as increasing the window size by the same amount gives even better results. Even a DFT and window size of 4096 usually introduces audible detuning for any scale factor.

Linear interpolation

With an overlap of at least 75%, linear interpolation clearly improves the result for window sizes up to 4096. The frequency resolution is much better in general, but other kinds of sound degeneration can occur. For window sizes up to 2048, modulation and bell-like artifacts are usually present in the resulting sound. With a window size of 4096, however, the result is virtually free of the disturbances even for larger scale factors (shifts about half an octave). The only issue left to deal with then is preserving the note attacks, which is not a matter of interpolation.

For a 50% overlap, the result is still in better tune than for integer shifts, though other frequencies are also audible close to the main peaks. For complex spectra with many peaks, this can be perceived as high-frequency amplitude modulation, a most undesirable disturbance that might not be preferred to the detuning effect of integer shifts.

Linear phase adjustment

The results of fractional phase shifts are comparable to those obtained from linear interpolation. Using settings for which linear interpolation works well there is usually little or no difference, and with other settings the result usually contains less artifacts and noise using fractional phase shifts.

For a 50% overlap, the disturbance found in the results from linear interpolation shifts is less audible.

The artifacts and noise occurring when using linear interpolation with at least 75% overlap and windows of 2048 or smaller are significantly reduced by fractional phase shifts.

When the phase shift is calculated from peak locations found by parabolic interpolation, the frequencies are slightly better preserved, and using phase unwrapping to find the peak locations gives a further minor improvement. This has been found from tests on square waves and is generally not noticeable for acoustic recordings.

4.1.4 Peak Detection

For uniform peak shifts of the entire signal, the definition suggested by Dolson and Laroche (1999), by which a bin is a peak if it is greater than its two neighbors on each side, usually gives the best result. Whereas the simpler definition, by which a bin is a peak when greater than one neighbor on each side, can give slightly better result for some inputs, the third – three neighbors on each side – usually performs much worse, and is not found to be better in any test.

When shifting individual peaks, the two-neighbor definition gives the best over-all result. However, when shifting peaks among the low frequencies, the region of influence is often too big, and to much of the spectral content is moved. In such cases, the one-neighbor definition is more suitable. Conversely, when shifting peaks at high frequencies the region of influence is sometimes to small, leaving relevant spectral information behind in the shift. This is less of a problem in this implementation, as a bigger region surrounding the peak can easily be selected. Nevertheless, the three-neighbor definition appears to be better suited for this part of the spectrum.

4.2 Applying the Operations

Here, the settings that produce the best results are tested together, and the range of useful operations is explored. The test data is also extended to include other instruments, and even ensemble recordings.

For the results presented here, the following settings are used:

- the square root Hann window as analysis and synthesis window
- linear phase adjustment (from integer bin peak position) for peak shift operations
- two-neighbor peak definition

The DFT, window and hop sizes are not fixed, as their optimal value greatly depends on the input and operation in question, and since this will determine the required amount of computation.

4.2.1 Uniform Peak Shifting

Scaling the spectral content uniformly can be done well in reasonable time for small scale factors. Larger scale factors will mainly require a smaller hop size to avoid loss of attack and general noise and artifacts. A DFT size of 2048 can be acceptable, but any less will surely result in noisy and inharmonic sounds. Therefore, satisfactory pitch shifts of about half an octave or more tend to be very time consuming, and the computations takes several times longer than the duration of the sound being processed.

For this kind of operation, with pitch shifts larger than a few semitones, the classical approach involving time scaling and resampling gives much better results. Using this technique, satisfactory sound quality at a reasonable computational expense can be found for shifts up to about one octave.

4.2.2 Single Peak Shifting

Single peak shifting is the most interesting function of the peak based vocoder, as it enables pitch shifting of single notes. Though other effects are also conceivable, such as changing the timbre or simply removing notes, there is no natural frame of reference for the evaluation of such modifications, and the tests are restricted to note shifts.

Peak selection for note shifts

To shift a given note, its fundamental frequency and a number of partials have to be located in the spectrogram. This can be tedious for music where many notes are played simultaneously. To change a note, it helps to have a good idea of what frequency it has and when it is played, as it can then be located by using the mouse pointer.

Once the fundamental frequency is found and shifted, the same shift has to be applied to some of its closest partials. How many depends on the harmony (and on the instrument/orchestra), but usually three or four partials will suffice to get a good result.

A commonly encountered complication arises when the note to be shifted shares some of its partials with other notes. In such cases, care has to be taken to change it only if the note being shifted is the stronger one, and to leave it otherwise. In some cases, due to such harmonic conflicts, there is no way to shift the desired note without introducing undesired artifacts.

It is often of interest to shift a note in the melody of some polyphonic recording, and melodies are usually played in higher frequencies than other notes. Therefore, the fundamental frequency of a note in the melody is often higher than the fundamentals of other notes, and even some of their stronger partials. In such cases, harmonic conflicts are less likely to have any major impact on the shifted result. Conversely, notes that have a low fundamental frequency have more closely spaced partials, and it is more likely that some of these will coincide with other prominent peaks.

Applicability

Shifting a single note can be sensitive to the transform, window and hop size, but not in the same sense as uniform peak shifts. As long as the fundamental frequency and partials of the note can be resolved, time and frequency resolution are of less importance to the quality of the result. Instead, the suitability of the settings depend on the spectrogram; to shift a peak that happens to be close in frequency to another peak, bigger transform and/or window size is necessary than in the cases when all peaks to be shifted are easily distinguished.

Any note that is well separated from near by notes in a spectrogram can usually be shifted. Good results are obtained for piano, guitar (acoustic and electric), organ and various synth sounds. As noted before, piano recordings will usually require a smaller hop size than many other instruments. For some inputs, perfectly satisfactory results can be obtained for a DFT and window size of 2048 and hop size of 512, or even a hop size of 1024 (giving 50% overlap). More generally, a transform size of 4096 and possibly a smaller window size, depending on the required hop size to preserve the note onset, will give good results for most inputs.

As to the range of shifts that can be performed, it is still limited by the fact that even a shift without artifacts or noise usually sounds unnatural when the shift is more than a few semitones. Piano and guitar notes can usually be shifted by 3-4 semitones, sometimes even half an octave. Organ notes are usually less sensitive to large shifts. For synth sounds it depends on the particular sound, and the highly subjective opinion of what might be expected after the shift.

The implementation of harmonic peak selection, whereby a scan is performed to look for any peaks located at the octave frequencies of the selected peaks, is not very reliable. One or a few harmonic peaks can sometimes be identified, but at a few partials distance from the fundamental peak, the accuracy is bad, and neighboring peaks are usually shifted instead.

Chapter 5

Conclusions

In the previous chapters, we presented an implementation of the peak based phase vocoder, and the results of some musical applications. The conclusions that can be drawn from this provide some insights into the possibilities of digital music processing. Furthermore, they point to possible improvements and give rise to some new questions for future research.

5.1 Results and Applicability

It is clear that the phase vocoder techniques tested here are very useful for a wide range of musical spectral modifications. In the evaluation of the results, two aspects should be considered. On the one hand, the result has to resemble a recording of the instrument in question and should not contain audible nose or artifacts that where not present in the original signal. On the other hand, the computations has to be completed in reasonable time. Here, reasonable time will be considered to mean that the processing time is less than the duration of the input signal, as this correspond to real time application in some sense.

Some general conclusions can be drawn about the settings that were tested. Among the windows that were implemented, the square root Hann and triangular windows gave the best overall results when used for both analysis and synthesis. This was expected for the Hann window; that the triangular window often performed equally well, and sometimes even better, was more surprising.

The two-neighbor peak definition was found to give the best result for general purposes. However, to select specific peaks in the spectrum, the other definitions were found to be more suitable for low and high frequencies, respectively. This implies that better results can be obtained if different peak definitions are used in different parts of the spectrum.

Perhaps the most interesting result regard the use of fractional phase shifts for interpolation. The simple linear interpolation implemented here improved the performance, though not as much as was expected. Adjusting the phase to correspond to fractional shifts comes at virtually no extra cost and gives even better result than linear interpolation. This can probably be explained by how simple linear interpolation of the amplitudes distorts the shape of the peak. In the worst case, the peak maximum is reduced by an amount determined by the height of the neighboring bins, and the shape of the peak can be changed quite unpredictably. With the linear phase approach, a peak is approximated by its exact shape, only slightly dislocated, which in the worst case gives a peak displaced by half a bin. The phase information is just as accurate as with linear interpolation.

When only integer shifts are allowed, Dolson and Laroche (1999) point out that the phase adjustment can be easily tabulated for most common overlap ratios. Compared to this solution, linear phase adjustment will require two extra trigonometric calculations per peak, but this is still cheaper than linear interpolation, which requires an additional complex multiply.

The results of uniform pitch scaling are satisfactory for small scale factors. For larger pitch shifts, the sound quality quickly degenerates, and to avoid this is simply too time consuming. It is possible that this can be helped by more sophisticated interpolation techniques. Nevertheless, it seems reasonable to conclude that

the real strength of the peak based approach lies in applications that can not be seen as linear transformations of the spectrogram. Here, pitch scaling by large factors would be more efficient with the standard technique, provided that efficient resampling is implemented.

We found pitch shifting of single notes to work very well, even when only a few partials are included in the shift. This is very interesting from a psycho-acoustic viewpoint, and also implies that automated note shifts – i.e. automated identification of relevant partials – might be relatively easy to obtain.

Results from polyphonic music transcription indicate that the task of classifying partials by which fundamental frequency they belong to can be quite challenging. However, to facilitate note pitch shifts, where the fundamental can be identified by the user, in many cases it seems to suffice to identify four or five partials to get good results.

The simple harmonic peak detection tested here gave poor results. One drawback with this implementation is that it scans for octave peaks only, instead of all multiples of the fundamental frequency. When two octave peaks can be identified, which is often the case, then one more harmonic peak should be identified if looked for, between the first and second octave peak. It is likely that a slight refinement to the implementation could then identify three octave peaks, and three more harmonic peaks between the second and third octave.

As for possible real time application, the existence of a lower limit to the transform size means that there is a lower limit to the latency between input and output, regardless of the time used for any computations. This means that real time application in sense that a sound is sampled, processed and then played back will involve a latency by at least 2048 sample points for mere sampling, which corresponds to almost 50 ms at the current sampling frequency (44.1 kHz). Furthermore, many results considered satisfactory here require at least half of the input signals duration in processing time, which point to a latency of 70-80 ms at best. This is a most noticeable latency for a musician and is unacceptable for any sound that has a distinct note onset. However, when timing is not an important issue, real time application might very well be useful.

Playback of some digitally stored music presents another real time application with even greater possibilities. For instance, listening to a piece of piano music it would be possible to mute or shift certain frequencies, including all octaves or harmonics, and controlling the frequencies to be affected in real time. In this case, the transform size is not an issue as the data can be read in advance, and only the processing time has to be considered, which greatly increases the possibilities.

5.2 On the Implementation

In the progression of this work, we found more literature and related results in the field that would have been interesting to consider. Some of these points deserves to be mentioned, and a few other points can be made about simple improvements that can result in much greater possibilities.

Optimal use of the FFT

The current implementation of the FFT takes no advantage of the fact that the input sequence is always real (this is only used at the highest level, in the phase vocoder). Algorithms exploiting this can save almost a factor of 2 in computation and memory, and it is easily accomplished by complex demodulation of the input signal (Elliot and Rao, 1982).

Optimal spectrograms

Phase vocoder processing of the spectrograms would probably benefit from the use of zero-phase windows, and the logarithm of the magnitudes for peak detection (Smith and Serra, 1987).

When using a zero-phase window, instead of deterministic windows as in the current implementation, the phase information of the signal is better preserved. This can give better preservation of note onsets, and better estimates of real frequencies when using phase unwrapping.

When using parabolic interpolation, the estimated true frequencies become about twice as accurate when using the logarithm of the magnitudes instead of linear magnitude. Though the current peak detection would not be affected, it is likely that more advanced harmonic analysis, such as note detection, would benefit from the use of logarithmic magnitudes. The fact that manual peak selection in the current implementation usually has to be done from logarithmic magnitudes suggests that it would also be esier to find useful spectral envelopes and threshold values.

Peak operations and note classification

The ability to not only shift peaks by a number of semitones or remove them, but to adjust their amplitude, copy them and apply finer pitch shifts, would make this application a much better tool for spectral sculpting. Apart from pitch shifting, other effects such as tuning, chorusing, harmonizing and more could be obtained.

With the ability to copy and change the amplitude of the peaks comes the ability to split peaks, possibly into components belonging to different notes. However, it would then be helpful to use a more refined classification of the peaks to obtain a better note detection.

One way to improve the note detection – i.e. the grouping of peaks into fundamental and partials – would be to look at all multiples of the frequency, and to apply some amplitude threshold to ensure that the partials are not louder than the fundamental frequency. One could then use some expected amplitude envelope, which, in case the instrument being processed is known in advance, could be quite detailed, or otherwise very general. If a partial should have a higher amplitude than expected, the peak could be split into two or more components, representing coinciding partials of different fundamentals, or, if no matching fundamental is found to account for it, a new fundamental itself.

5.3 Afterwords and Future Research

The work presented in this thesis began with the study of the fundamentals of digital signal processing and the Fourier transform. The aim was loosely to investigate the possibilities of musical analysis, and the final formulation of the thesis was decided only when the implementation process had begun, and the work of Dolson and Laroche (1999) was discovered. This has resulted in a rather wide theoretical scope, where many different questions have been touched upon, and the result is a summary of the main issues that has to be considered when using phase vocoder techniques to achieve musical pitch modifications.

As for the results, it can be pointed out that the evaluation is quite subjective. Ideally, the results should be shown to a larger group of people, and a mean of the perceived quality should be used for evaluation. However, the time and effort required for such an arrangement can be better invested, as many implementation issues are yet to be considered.

The front of present research in the field have become more familiar in the progress of this work. In retrospect, it would have been interesting to give more focus to the issues of interpolation and peak selection. One question that is posed by Dolson and Laroche (1999), that appears to be unanswered still, is whether one might benefit from the use of more advanced interpolation methods, such as Lagrange polynomials.

Other interesting questions for future research can be found by turning to the fields of polyphonic transcription and voice synthesis. A major issue in polyphonic music transcription is how to factorize the spectra into different notes. Applying such analysis would make the application easier to use, and require less understanding of music theory from the user.

Results from voice synthesis of great interest to the present task is how to detect formants, and how to preserve spectral envelopes when shifting the pitch. An incorporation of these results should help to preserve the instrument characteristics when applying large pitch shifts.

Bibliography

Bennewitz, C. (2007). Fourier analysis. Literature for the course Fourier Analysis at Lund University, fall 2007.

Bourke, P. (1996). Creating TGA image files. `http://local.wasp.uwa.edu.au/~pbourke/dataformats/tga/`. accessed 2010-03-01.

Dolson, M. (1986). The phase vocoder: A tutorial. *Computer Music Journal*, 10(4):14–27.

Dolson, M. and Laroche, J. (1999). New phase-vocoder techniques for pitch-shifting, harmonizing and other exotic effects. In *Proc. 1999 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 91–94, New Paltz, New York.

Elliot, D. F. and Rao, K. R. (1982). *Fast Transforms: Algorithms, Analyses, Applications*, chapter 3. Academic Press, New York.

Flannery, B. P., Press, W. H., Teukolsky, S. A., and Vetterling, W. T. (2007). *Numerical Recipes: The Art of Scientific Computing*, chapter 12. Cambridge University Press, New York, 3 edition.

Hayes, M. H. (1999). *Shaum's outline of theory and problems of digital signal processing*, chapter 7, pages 262–266. McGraw-Hill.

Sapp, C. (2003). WAVE PCM soundfile format. `https://ccrma.stanford.edu/courses/422/projects/WaveFormat/`. accessed 2010-03-01.

Smith, J. O. (2009). Spectral audio signal processing, march 2009 draft. `http://ccrma.stanford.edu/~jos/sasp/`. accessed 2010-03-01.

Smith, J. O. and Serra, X. (1987). PARSHL: A program for the analysis/synthesis of inharmonic sounds based on a sinusoidal representation. Technical Report STAN-M-43, CCRMA, Department of Music, Stanford University. `https://ccrma.stanford.edu/STANM/stanms/stanm43/stanm43.pdf`.