Pitch Shifting using STFT

Hao Yu, hy987, N15474757

Pitch shifting can be realized by simply changing the frame rate when playing the sound. But in this way, the length of the signal is also changed. So if we can scale the time of the signal while leave the pitch unchanged, we can get a pitch shifted version of the original sound with the same length.

Time Scaling

Therefore, the most important part in pitch shifting becomes time scaling. To scale the time we can use different hop sizes when taking the STFT and inverse STFT. While the step size is scaled, the window size is identical. But we cannot just add the frames up in reconstruction. To reduce the discontinuity, we need some spectral processing. This technique is called phase vocoder. It consists of 3 stages: analysis, processing and synthesis.

In analysis, we use STFT to get the spectra of the overlapping frames of original signal. In processing stage, we do some spectral processing. And in synthesis, the processed frames is inverse transformed to time domain and overlap.

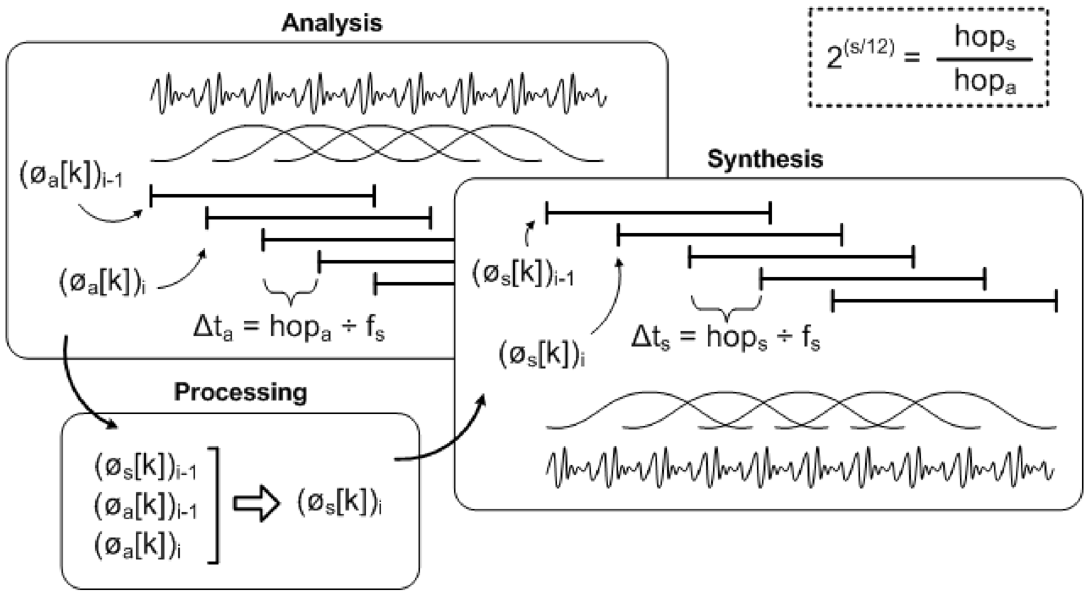
Analysis
$2^{(s/12)} = \frac{hop_s}{hop_a}$
Synthesis
$(\varnothing_a[k])_{i-1}$
$(\varnothing_a[k])_i$
$\Delta t_a = hop_a \div f_s$
$(\varnothing_s[k])_{i-1}$
$(\varnothing_s[k])_i$
$\Delta t_s = hop_s \div f_s$
Processing
$(\varnothing_s[k])_{i-1}$
$(\varnothing_a[k])_{i-1}$
$(\varnothing_a[k])_i$
$(\varnothing_s[k])_i$

Figure 1 Phase vocoder overview

Phase Correction

In frequency domain, a sound wave is interpreted as the magnitudes and phases of the frequencies. We usually don't modify the magnitudes of the frequencies, as the magnitude represent the energy of a frequency component. So we only correct the phase.

The frequency is the rate of change of the phase. Once we know the frequency, we can calculate the phase change. When taking DFT, we get regularly spaced discrete frequency components, namely bin frequencies.

$$\omega[k] = \frac{2\pi k}{N} \ , k = 0,1,2,\ldots,N-1$$

To deal with a sinusoid fitting perfectly into the bins is easy. The phase change is proportional to the bin frequency. We just need to multiply the bin frequency with the hop size.

$$\Delta\emptyset_{expected}[k]_i = \omega[k] \times hop$$

But for the signal in the real world, there will always be spectral leakage. The leakage will affect some adjacent bins. So there will be some deviation from the expected phase change in the affected bin. In figure2, the solid target represents the expected frequency change while the dashed one denotes the actual phase change contributed by the leakage. With the deviation, we can derive the instantaneous frequency.

$$(\omega_{true}[k])_i = \frac{(\Delta\emptyset_a[k])_i}{\Delta t_a}$$

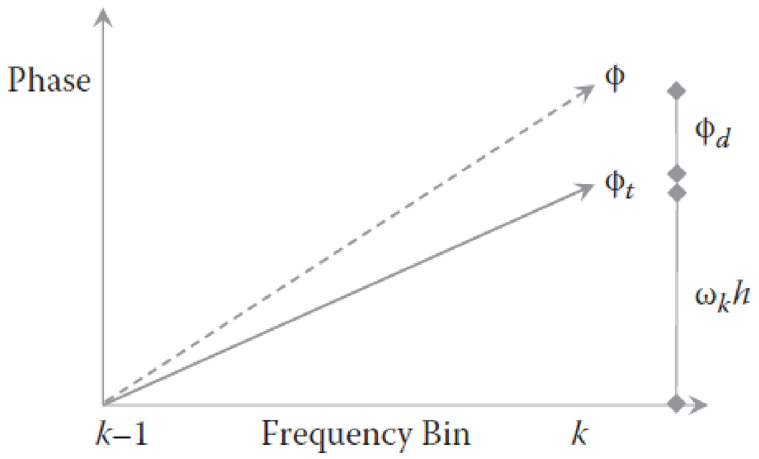

Figure 2

And with true frequency, we can come up with the phase deviation in synthesis hot size.

$$(\Delta\omega_s[k])_i = \frac{(\emptyset_a[k])_i - (\emptyset_a[k])_{i-1}}{\Delta t_a} - \omega[k]$$

There's one more step to go. The true frequency needs unwrapping to make sure it's fall in the interval from –π to π.

$$\left(\Delta\omega_{wrapped}[k]\right)_i = mod[((\Delta\omega[k])_i + \pi), 2\pi] - \pi$$

Now we derive the true frequency:

$$(\omega_{true}[k])_i = \omega_{bin}[k] + \left(\Delta\omega_{wrapped}[k]\right)_i$$

The phase change in the synthesis frame is

$$(\phi_s[k])_i = (\phi_s[k])_{i-1} + \Delta t_s(\omega_{true}[k])_i$$

The corrected phase is then combined with the magnitude and transformed back to time domain for overlapping.

Phase Locking

In the phase correction process, there may be some frequencies near to each other, so the bins between them will be affected by both. In this case, the estimate of the true frequency may not be accurate.

Phase locking is a technique to reduce the artifacts resulting from this case. First, we locate the prominent peaks (the ones nearest to the true frequencies) by detecting the local maximums. And define the region of influence. For all the bins in the region, the phase change is the same as that of the peak.

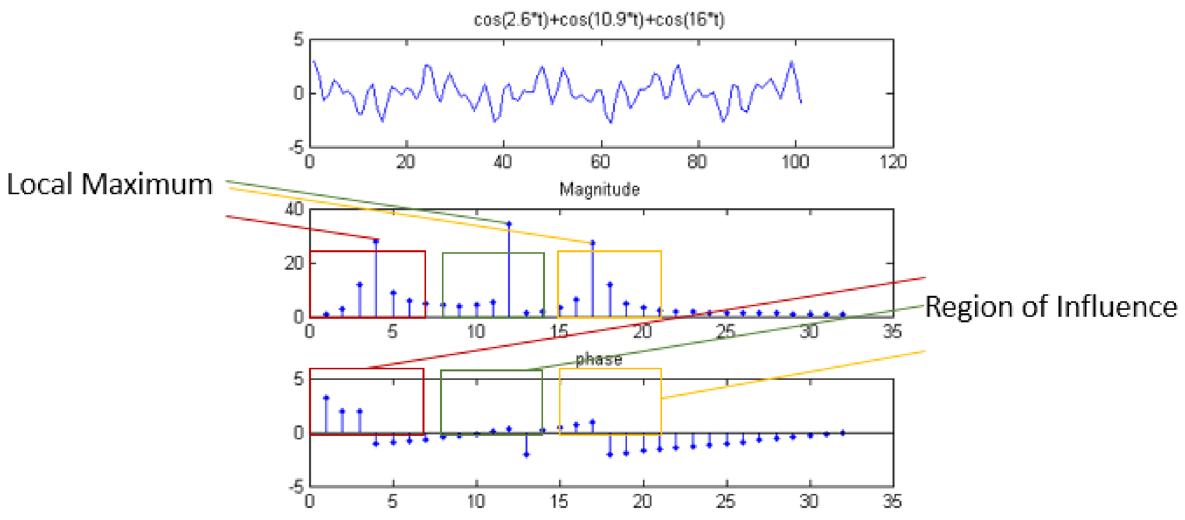

Figure 3

Resampling

Now after synthesis, we have a time scaled version of the original sound with unchanged pitch. To shift the pitch, you can simply change the playback rate. But in realtime processing, we usually resample the sound. In my implement, the resample is realized with linear interpolation.

$$y[n] = (1 + \lfloor Rn \rfloor - \lfloor Rn \rfloor)x[\lfloor Rn \rfloor] + (Rn - \lfloor Rn \rfloor)x[\lfloor Rn \rfloor + 1]$$

x is the time scaled signal and y is the resampled signal. R is the resample factor.

Implementation

In my implementation, the window size is fixed to 2048 and synthesis hop size 512.
The size of the block of data is 4 times analysis hop size.
In peak detection, the boundary of the region of influence is set to the middle point of two adjacent peaks. For the first peak, the region starts from 0. For the last peak, the bins after the peak is ignored.
In non-realtime version, the signal is resampled after the signal is processed.
In realtime version, when the buffer is filled with Fs/pitch_ratio/4 samples, the data is written out. Frequent write operations incur some artifacts while larger chunks introduce more delay.
The GUI is based on pygame. The label, button and scroller is from the internet (see reference 7) and I modify the button by adding the press.

Reference

1. Florian Hammer, “Time-scale Modification using the Phase Vocoder”
2. Joshua D. Reiss, Andrew P. McPherson, “Audio Effects Theory, Implementation and Application” Chapter 8 Phase Vocoder.
3. Pitch Shifting Using The Fourier Transform
   http://blogs.zynaptiq.com/bernsee/pitch-shifting-using-the-ft/
4. Guitar Pitch shifter
   http://www.guitarpitchshifter.com/algorithm.html#33
5. A phase vocoder in python
   https://audioprograming.wordpress.com/2012/03/02/a-phase-vocoder-in-python/
6. Python, Pitch Shifting, and the Pianoputer
   http://zulko.github.io/blog/2014/03/29/soundstretching-and-pitch-shifting-in-python/
7. miniGUI
   http://cs.iupui.edu/~aharris/pygame/ch09/miniGUI.py