An Application of STFT

Time Stretching and Pitch Shifting

This paper illustrates an application of the Short Time Fourier Transform (STFT) by using Phase Vocoder to time stretch and pitch shift audio signals, also implying on it's use in the audio digital signal processing.

Ayan Shafqat

12/14/11

An Application OF STFT

Time Stretching and Pitch Shifting

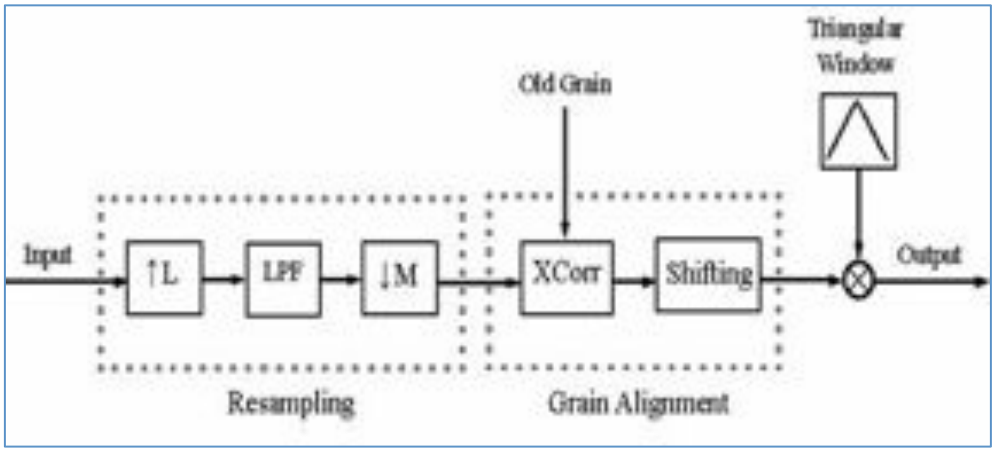
INTRODUCTION

Time Stretching and Pitch Shifting is widely used by audio professionals all around the world due to various benefits it provides. First of all, it commonly found that DJ's often want to remix songs in different tempo than the original. For that purpose, he or she may have two options present. On one side, the DJ can play that segment slower, similar to the "speed" knob on an old tape recorder. This solves the problem, but it also changes the overall pitch of the audio signal, which the DJ may not want. There is another option for DJ's now, which is commonly known as time stretching. This allows a DJ or anyone to change the tempo of a song, without compromising the pitch attributes of the signal.

Similarly, an audio engineer may want to correct a singer's pitch without doing another retake in the studio. Since studio time is costly, pitch shifting allows the engineer to correct the pitch of a singer's voice to his desired levels, without compromising on the duration of the signal. A very elaborative application of this known as a pitch corrector, more commonly known as Auto-Tune, which evolved from the product's name of the company Antares. The purpose of this application was to detect the pitch of the singer and automatically shift the detuned pitch into the nearest quantized note. Sometimes, it is also used as a vocal effect, used by many artists, such as Eiffel65, Cher, T-pain, among many others.

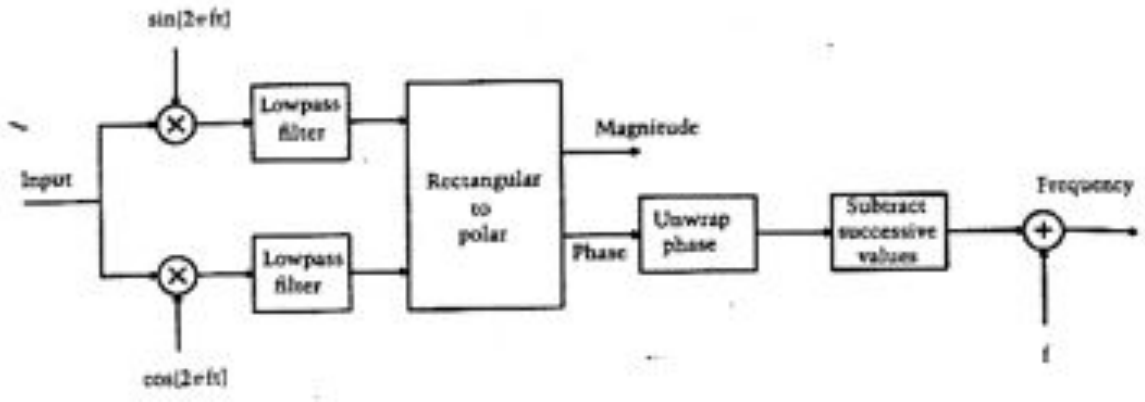
Antares AutoTune: A world-class pitch correcting software (http://www.antarestech.com/products/auto-tune-evo.shtml)

There are many methods of shifting and stretching an audio signal. Some of them time domain methods including, Sample Overlap and Add (SOLA), Pitch Synchronous Overlap Add Method (PSOLA). These methods work well for monophonic sounds, but fail to work well for polyphonic signals. This is mainly due to the fact they need to find the fundamental frequency of the signal to change the length of the signal. For example, if a cycle of sine wave at a certain frequency f_0 needs to be shifted by a factor of two, we can assume that we can just copy the signal twice, and down sample by a factor of two. But, having inharmonic signals present in that audio signal will result into an unpleasant sound. Therefore, it is often observed that formants are lost when pitch shifting with these time-domain techniques, resulting into the famous "chipmunk" sound.



Signal diagram of SOLA
(http://homepages.inspire.net.nz/~jamckinnon/report/sola.htm)

Another form of time stretching and pitch shifting can be done with interpolation in the frequency domain, which is known as Phase Vocoder (Flanagan J.L. and Golden 1966). Its name derived from the original “vocoder,” which was a contraction of “voice encoder,” was first used in encrypting voice over telephone during World War II. The original vocoder divided the signal into different parallel band-pass components, and then resynthesized using oscillators. This allowed the signal to be modified without affecting the overall speech. Similarly, applying phase vocoder in a signal resulted into a signal which is time stretched by a factor, without compromising pitch information. The diagram below shows one of the band-pass filter components and it’s re-synthesis unit.

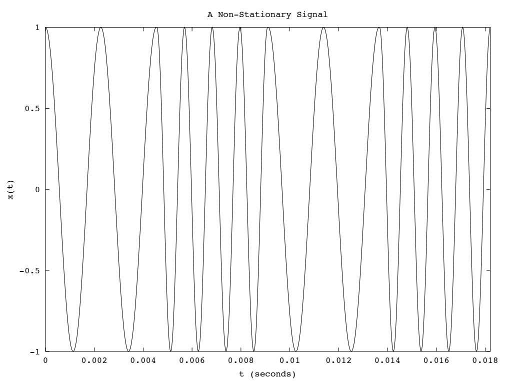


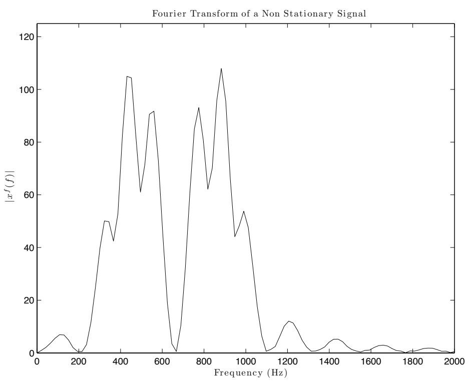
Signal Diagram of one filter-bank component of a Phase Vocoder
(http://www.panix.com/~jens/pvoc-dolson.par)

When we consider digital signal processing, it is a lot more efficient to implement this with Fast Fourier Transform (FFT), since FFT can be thought of as N number of band–pass filters. Also, FFT is invertible, for which re-synthesis can be done in an efficient manner as well. But, an audio signal of our interest is very rarely stationary, for which we can use Short Time Fourier Transform. In the next sections, we will explore how to implement time–stretching and pitch–shifting algorithms using STFT.

Short Time Fourier Transform (STFT)

Consider this non-stationary signal shown below. There are 800 samples of two interchanging frequencies. Taking the Fourier Transform will result into peaks in two different frequencies (in this example, 441 and 882Hz). We certainly know that this is not the case in this signal, for which we need to consider STFT.





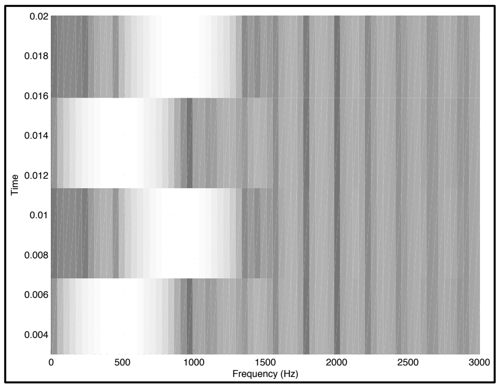
STFT, as the name sounds, takes a Discrete Fourier Transform in different short segments of a signal. This takes account for the frequency domain, which changes in time. Thus, taking the STFT of a signal will result into a two–dimensional matrix consisting of vectors of the DFT coefficients. Therefore, STFT is not a frequency domain or time domain representation of a signal, but both joint time-frequency domains, which can be very useful when analyzing non–stationary signals. Each DFT vectors are known as a frame (denoted as m), and the index of the coefficients is called the bin (denoted as k). STFT is defined as:

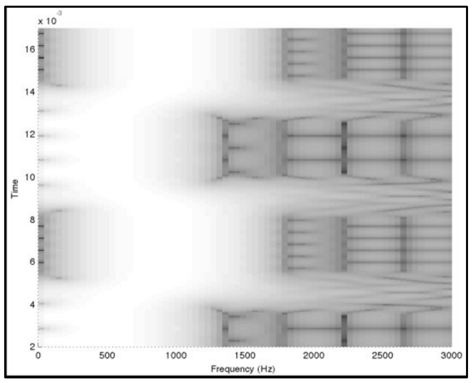
$$X(\omega, m) \coloneqq DTFT\{x(n-m)w(n)\}$$

$$X(\omega, m) = \sum_{n=\infty}^{\infty} x(n-m)w(n)e^{j\omega n}$$

$$\approx X[k, m] = DFT_N\left\{[x(n-m)w(n)]_{n=0}^{R-1}, \mathbf{0}_{N-R}\right\}$$

When we calculate STFT, there are several parameters we must consider. First of all, we would want to consider the block length we are considering, denoted as R. But, in order to increase its accuracy, we must consider the overlap factor as well, denoted by W. The figures below show the result of doing an overlap vs. no overlap. Note the increase in resolution as we increase the overlap.





Semi–Perfect Reconstruction Condition of STFT

In order to make STFT invertible, we have to follow a strict rule for the window design that satisfies the condition below. This is known as the perfect reconstruction condition.

$$\sum_m w^2\left(n - m\frac{R}{W}\right) = 1$$

But, sometimes this condition cannot be fulfilled due to different obstacles. Especially when we are changing the DFT coefficients, or changing the overlap size, there are problems satisfying this condition. Therefore, we just need a reconstruction condition, which is known as a Semi-perfect reconstruction, which is shown below.

$$\sum_m w^2\left(n - m\frac{R}{W}\right) = \alpha$$

In this condition, we do not care if the signal matches the original. But as long as they converge to a constant, we should not hear any artifacts in the modified signal.

Time Stretching

As said before, time stretching allows us to stretch the signal without changing the frequency contents of the signal. An easy way to achieve this would be to take the STFT, and change the overlap size. But, this will not result into a pleasing sounding audio signal, since there is a presence of phase discontinuity. We can think of a signal in terms of the magnitude and phase, which we would want to keep linear. From the STFT section, we can say that, $X[k,m]$ can be divided into $|X[k,m]|$ and $\measuredangle X[k,m]$. Also, we would want to model the signal such that, $\measuredangle X[k,m]$ is a straight line that starts at the initial phase and goes down with a constant slope. Therefore, we need to modify the phase to keep it continuous. From the phase vocoder diagram that was presented before (shown in page 2), by unwrapping the phase and subtracting from the previous phase, we can make the phase continuous. Thus, we can eliminate phase discontinuity and make the signal sound more natural.

What is a good length, and hop size? How should we choose hop size vs. sample frequency?

The compromise that we have to make when implementing phase vocoder is to choose the best STFT length according to the sample rate. If we choose a low length, then it does a better job at stretching "impulsive sounds" such as drums, but fails at stretching melodic sounds such as a piano. Also if we choose a large length, then the opposite happens, where the drums get very distorted, while melodic components sound very clean.

Choosing an initial hop size is very important in stretching or compressing the signal. If we are stretching the signal, then we would want to start with a low hop size, since the resulting hop size will be larger. Conversely, if we would want to compress a signal, then we would want to choose a larger hop size, since the resulting hop size will be smaller. Therefore, these formulas are proposed in this paper.

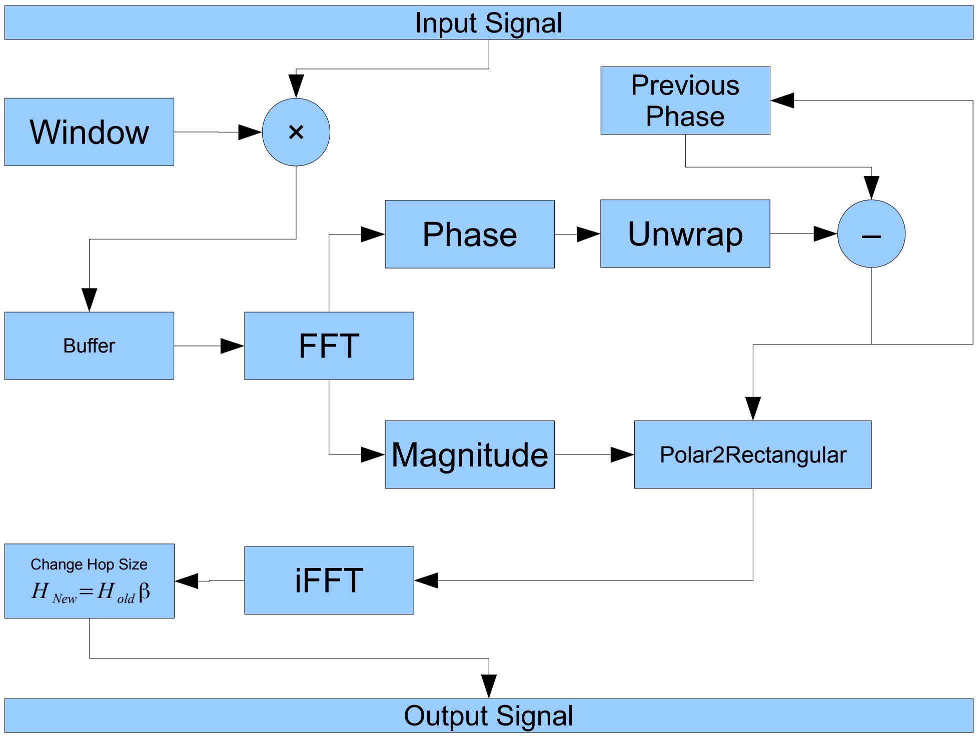
$R = \left\lfloor \frac{k}{\beta} f_s \right\rfloor$, where k is a constant of 25mS. Overlap Factor is: $W = \frac{N}{2^{\lceil \beta \rceil}}$

Zero Padding:

In order to maintain the optimal phase for the vocoder to function, we align the signal in the center of the zero padding of the FFT (Lim June, 2007). Suppose, we have an R length signal, we want to zero pad the signal to the next power of two, N. Then:

$$x_z = \left\{ \mathbf{0}_{\frac{N-R}{2}},\ \ x,\ \ \mathbf{0}_{\frac{N-R}{2}} \right\}$$

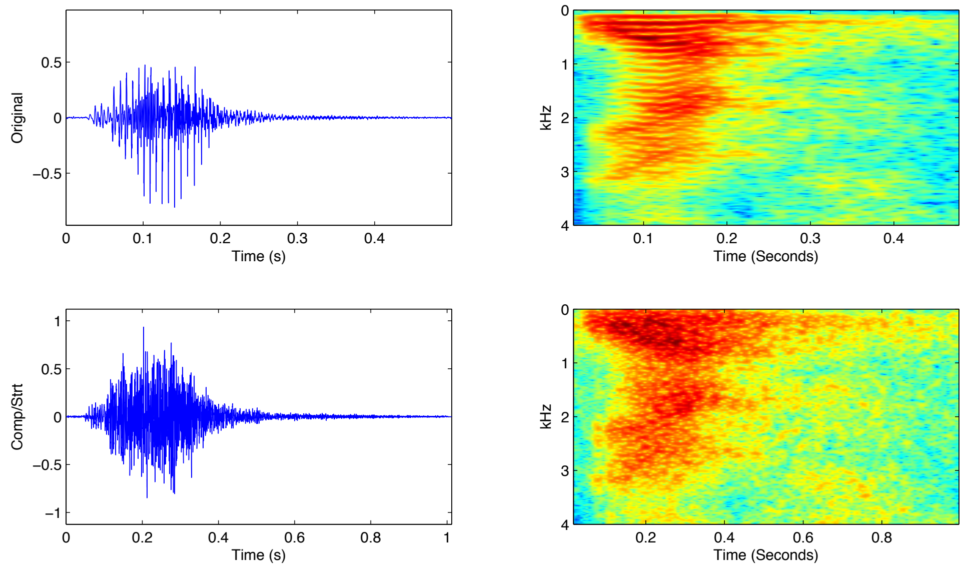
The overall signal chain of time stretching algorithm is shown in the figure below.



MATLAB CODE FOR TIME STRETCHING

```
function y = timeStretch(x, fs, r)
% ---------------------------------------------------------------------------
% timeStretch allows to stretch a signal without modifying frequency
% contents of the original signal, using short time fourier transform
% (STFT).
% Inputs:
%      x  = original signal
%      fs = sampling frequency
%      r  = factor (r < 1 means stretch signal and r > 1 to compress)
%           r cannot be less than or equal to 0
% Outputs: y = time stretched signal
% ---------------------------------------------------------------------------
    if(nargin ~= 3), error('Usage: y = timeStretch(x, fs, r)'); end;
    if(r <= 0), error('Stretch factor (r) has to be greater than zero'); end;
    r = 1/r;
    % Step 1: Creating a window that has perfect reconstruction condition at
    % N/4 overlap. This would be a half-cycle sine wave
    % Length of window is determined by 30mS worth of data
    K = 25e-3;
    N = round(K * (1/r) *fs);
    if(r > 1), overlapDenom = (2^ceil(r));
    else       overlapDenom = round(4*r); end
    if(mod(N,overlapDenom) ~= 0),
       % Making sure N is a multiple of overlapDenom
       N = floor(N + mod(N,overlapDenom)).*overlapDenom;
    end
    window = genSinWindow(N);
    % Step 2: Pre–calculation
    NFFT = (2^nextpow2(N))*4; % FFT Length with zero padding
    hop  = N/overlapDenom;
    mHop = round(hop.*r); % Modified hop after time stretch
    L    = length(x);     % Signal length
    NFr  = ceil(L/hop);   % Number of frame
    x    = x(:);          % Making sure x is a column vector
    % Zero padding x to prevent dimesion overflow
    x    = [x; zeros(ceil((NFr*hop + N) - L), 1)];
    L2   = NFr*mHop + N; % New length of signal
    y    = zeros(L2, 1); % Output signal
    n1   = 0;
    n2   = 0;
    phasePrev = fft(fftshift(padZerosAlignMiddle(x(1:N), NFFT)));
    phasePrev = phaseUnwrap(phasePrev,fs);
    while (n1 < L),
        xTemp = x(n1 + (1:N)) .* window;
        xTemp = padZerosAlignMiddle(xTemp, NFFT);
        Xk    = (fft(fftshift(xTemp)));
        n1    = n1 + hop;
        mag   = abs(Xk);
        phase = phaseUnwrap(Xk, fs).*(hop*r);
        phase = phase - phasePrev;
        phasePrev = phase;
        Xk    = mag .* exp(1i .* phase);
        xTemp = fftshift(real(ifft(Xk)));
        y(n2 + (1:N)) =  y(n2 + (1:N)) + xTemp(1:N).*window;
        n2    = n2 + mHop;
    end
    [b a] = butter(4, 20/fs,'high');
    y = filter(b, a, y);
    y = y ./ (max(abs(y)) + 10e-3);
end
function theta = phaseUnwrap(C, fs)
    phi   = atan2(imag(C), real(C));
    theta = zeros(size(phi));
    N     = length(phi);
    theta(1) = phi(1).*(fs/(2*pi*N));
    for k = 2:length(phi)
       theta(k) =  (phi(k) - phi(k - 1)).*(fs/(2*pi*N));
    end
end
function y = padZerosAlignMiddle(x, N)
    x = x(:);
    if(length(x) > N),
        error('Zero padding error');
    end
    L = length(x); M = N - L;
    if(mod(M, 2) ~= 0),
        K = (M-1)/2; Z = zeros(K,1); y = [Z;x;Z;0];
    else
        K = M/2; Z = zeros(K,1); y = [Z;x;Z];
    end
end
function window = genSinWindow(N)
    n = 0:(N-1); window = sin(pi./N * ( n + 0.5) ); window = window(:);
end
```

The output of this algorithm is shown below. Note the signal is twice as long, but the frequency content remains the same for both signals.



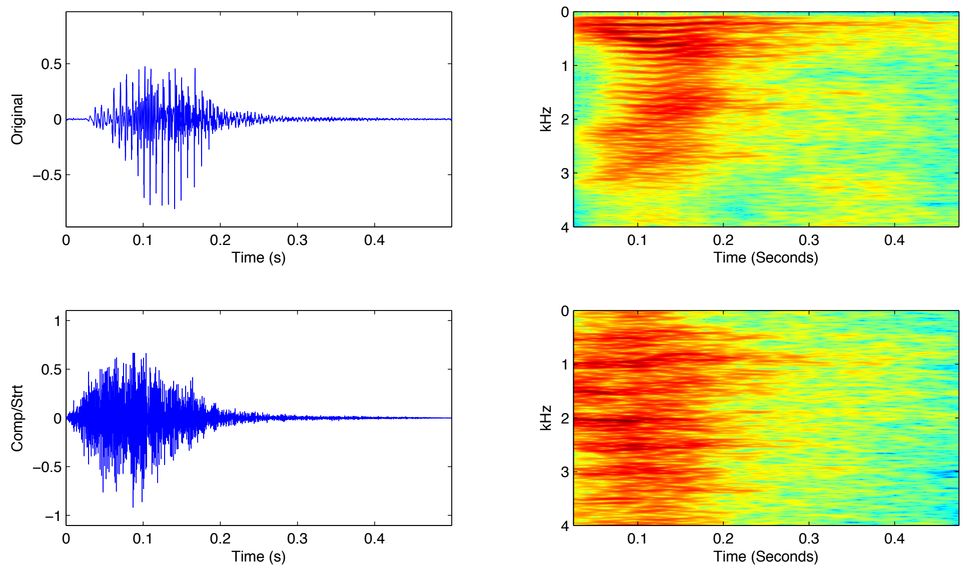
Pitch Shifting

Pitch shifting is actually a two-step process using phase vocoder. At first we stretch the signal with a factor, which is inverse of the factor we would want to pitch shift by. Then, we resample the signal to the original length. The code for this is shown below.

MATLAB Code for Pitch Shifting

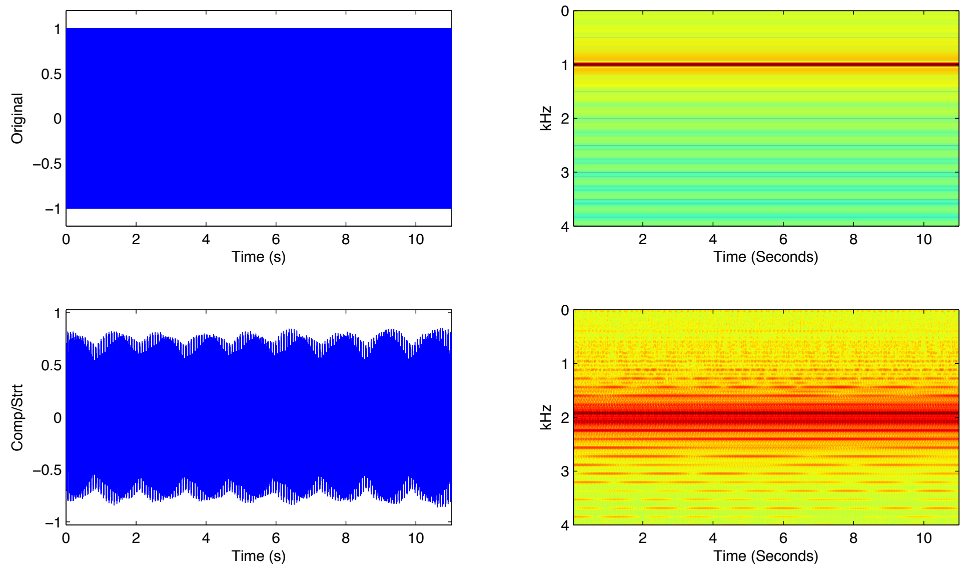
```
function y = pitchShift(x, fs, semitone)
% ---------------------------------------------------------------------------
% pitchShift allows to change pitch of a signal without modifying time
% contents of the original signal, using Short Time Fourier Transform
% (STFT).
% Inputs:
%      x   = original signal
%      fs = sampling frequency
%      Semitones   = number of semitones
%          ^- http://en.wikipedia.org/wiki/Semitone
% Outputs: y = pitch shifted signal
% ---------------------------------------------------------------------------
if(nargin ~= 3),
        error('Usage: y = pitchShift(x, fs, semitone)');
    end
    r   = 1/(2^(1/12 *semitone));       % time stretching factor
    y   = timeStretch(x, fs, r);
    t1 = (0:(length(y)-1))./fs; t2 = t1/r;
    y   = interp1(t1, y, t2, 'spline'); y = y(:);
    if(length(y) > length(x)), y = y(1:length(x));
    else
        y = [y; zeros(length(x) - length(y),1)];
    end
end
```

The output of this is shown below. Note the fact that, the signal is in the same length, but the frequency contents has shifted by a factor of two.



Some Issues with Phase Vocoder

As shown below, when we input a stationary signal into the phase vocoder (1KHz sine wave), we would expect a single peak at twice its frequency (considering the pitch shift factor is 2). But, due to window overlapping, it does not reconstruct a stationary signal.



Conclusion:

In order to create a more robust phase vocoder, there have been several articles that have proposed the use of Wavelets. But, due to today's computation limits, it may not be widely available in real time. There have been a few proposals that use both time and frequency domain shifting methods that showed promising results. Many products such as Eventide's Pitch Factor use this technology to produce rich timbre.

Eventide Pitch Factor
http://www.eventide.com/AudioDivision/Products/StompBoxes/PitchFactor.aspx

Bibliography

Flanagan J.L. and Golden, R. M. "Phase vocoder." *Bell System Technical Journal* 45 (1966): 1493-1509.

Lim, Kyung Ae. "An Open-Source Phase Vocoder with Some Novel Visualizations." Capstone Project, School of Informatics, Indiana University Bloomington, Bloomington, Indiana, June, 2007.