VIBRATO DETECTING ALGORITHM IN REAL TIME

Minhao Zhang, Xinzhao Liu

University of Rochester Department of Electrical and Computer Engineering

ABSTRACT

Vibrato is a fundamental expressive attribute in music, especially in singing, in stringed instrument performance, and in performance techniques of many wind instruments. Performers typically invest a great deal of time and practice to gain adequate control of vibrato in performance. To assist and accelerate this learning process, we plan to develop a computer based vibrato visualization tool.
In this paper we will explore two ways to detect the vibrato based on preceded research work. We coded each of the methods and tested them. Then we try to apply the algorithm in our real time visualization tool. The target result tries to show the FM and AM information on the screen using some trajectory.

Index Terms— One, two, three, four, five, six

1. INTRODUCTION

In this paper, we summarized a clearer definition of vibrato, both from its perception standing and from digital signal processing standing. Then we explored the features of vibrato, the main factors cause the music vibrato effect in section two. In section three, we proposed two algorithms fit for detecting the features of vibrato in real time, which are amplitude modulation (AM) and frequency modulation (FM). Based on the algorithms, we developed a real time vibrato visualization software written in C/C++ to show the FM and AM in music vibrato. Section five will explain the basic structure and libraries we used to build this program. In section six, we will summarize the defects, bug and some unfinished parts in this project, which then will lead to the future work we will keep working on.

2. DEFINATION OF VIBRATO

Vibrato is generally defined as a vibrating quality related to pseudo-harmonic modulations of pitch, intensity or spectrum which alone or in combination serve to enrich the timbre of musical sounds. This vibrating of pulsating aspect of vibrato can be attributed to as least one of the tree components:
1. Fundamental frequency pulsations which are perceived as pitch pulsation, which is the key term—FM we mentioned above.
2. Intensity pulsations which are perceived as loudness pulsations, which is the AM term mentioned above.
3. Spectral enrichment cycles which correspond to spectral envelope pulsations. This can be perceived as brightness modulation.
But in this paper, our algorithm will not detect spectral envelope pulsation, so no further discussion will be placed in the rest of the paper. More work on spectral envelope pulsation can be found in the paper of Verfaille *et al.,* 2005.
In signal processing, AM feature of Vibrato can be described as (McAulay & Quatieri, 1986, Serra & Smith, 1990):

$$\boldsymbol{x(n) = \sum_{h=1}^{H} a_h(n) \times cos(\Phi_h(n))} \qquad (1)$$

AM is $a_h(n)$ in equation (1). h represents the harmonic number that a music note has. $x(n)$ is the final waveform.
The phase is given as the integral of time-varying frequency $f_h(n)$:

$$\boldsymbol{\Phi_h(n) = \Phi_h(n-1) + 2\pi + \frac{f_h(n)}{F_s}} \qquad (2)$$

F_s is the sampling frequency and $\Phi_h(0)$ is the initial phase. The FM feature of the vibrato is then the $f_h(n)$ in equation (2). We can see from the AM and FM terms, they both time varying expression, which explains the modulation term in vibrato.

3. AM AND FM DETECTION

The core of this project in theory is AM and FM detection algorithm. This section will first discuss the algorithm we performed in our code to detect the AM and FM in music vibrato. In our program, it is a real time processing unit, but in this section, we will use the signal as an offline one channel signal to explain the algorithm.
For an offline signal, we perform the following steps to detect the amplitude modulate and frequency modulation. In amplitude modulate we will just detect the instantaneous amplitude of the fundamental frequency. In frequency we will detect both the modulation depth—vibrato depth and modulation frequency—vibrato rate. In our algorithm, we assumed the signal is relatively clean without much noise. We also assumed that the offline signal is a harmonic note. So we do not need to apply any filtering before we process the signal or to differentiate if it is a voiced or unvoiced signal.

Step 1: Preprocessing:
After getting the raw signal, we remove the DC part of the signal by subtracting the average power. Then we perform a short time Fourier transform on the signal. We compute the

global peak value and then we normalize the signal so the highest harmonic peak will be 1.

Step 2: Locate the fundamental frequency bin:

For each frame in our frequency domain, we do a linear search to find the fundamental frequency bin. In order to accurately find it, we set a threshold for the peak amplitude to be 0.1 and the minimal fundamental frequency to be 50Hz. But his value can be changed. Then the first local max that satisfies the threshold above will be our fundamental frequency. Figure 1 shows an example of peak finding with threshold of 0.1. We can see for a clear harmonic sound, we can find the fundamental harmonic of the note, since fundamental frequency is usually on the strongest peak in a note.

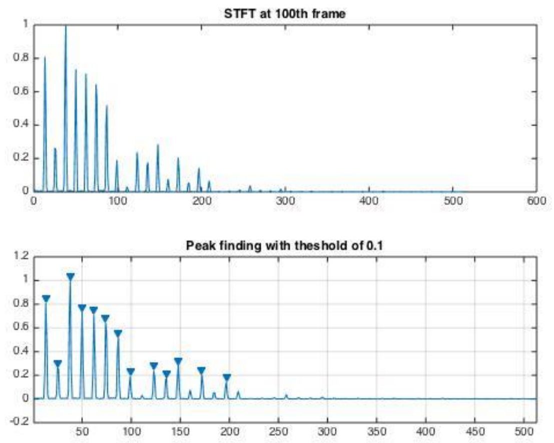

Figure 1

Step 3: Instantaneous frequency and amplitude detection:

In this step, based on the fundamental frequency bin we had located, we will perform the FM depth estimation, which is also called the instantaneous frequency detection. And we used two methods to accomplish this task

Method 1:

This method is also called time frequency processing (Zoler, 2011). As this name implicated, a short time Fourier transform will firstly applied to the signal, which is done in our first 2 steps. Then for the fundamental peak bin, we call it bin k at time n. n represents our time index in frame number. We can can say the amplitude of this bin k at n is $\alpha(n,k)$ then the phase of it is $\phi(n,k)$.

For AM detection, the amplitude of each frame $\alpha(n,k)$ is our AM detection.

Now for FM detection, it is more complex than AM detection. We need to calculation of an unwrapped phase:

$$\boldsymbol{\tilde{\Phi}(n,k) = \frac{2\pi k}{N} + \Phi(n,k)} \qquad (3)$$

The phase computation are based on the phase values $\tilde{\Phi}(sR_a,k)$ and $\tilde{\Phi}((s+1)R_a,k)$, which are the results of FFT of two consecutive frames. R_a is the time difference between the two consecutive frames. Our goal is to calculate the instantaneous frequency of each frame, which is our frequency modulation. The followings are the steps:

1. Calculate the targeted phase $\widetilde{\Phi_t}((s+1)R_a,k)$ based on the previous phase value $\tilde{\Phi}(sR_a,k)$.
2. The unwrapped phase will be:

$$\boldsymbol{\widetilde{\Phi_u}((s+1)R_a) = \widetilde{\Phi_t}((s+1)R_a,k) + \widetilde{\Phi_d}((s+1)R_a,k)} \qquad (4)$$

3. The term $\widetilde{\Phi_d}((s+1)R_a,k)$ means the deviation phase, which can be calculated as following:

$$\widetilde{\Phi_d}((s+1)R_a,k) =$$
$$\boldsymbol{wrap[\tilde{\Phi}((s+1)R_a,k) - \widetilde{\Phi_t}((s+1)R_a,k)]} \qquad (5)$$

4. Now combine equation (4) and (5), we can derive the unwrapped phase difference:

$$\Delta\Phi((s+1)R_a) =$$
$$\boldsymbol{\omega_k R_a + wrap[\tilde{\Phi}((s+1)R_a,k) - \widetilde{\Phi_t}((s+1)R_a,k)]} \qquad (6)$$

5. So the instantaneous frequency for frequency bin k at time instant $(s+1)R_a = n$ is

$$\boldsymbol{f_i((s+1)R_a) = \frac{\frac{1}{2\pi}\Delta\Phi((s+1)R_a)}{R_a} f_s} \qquad (7)$$

To test this method, we synthesized a pure sinusoidal tone at pitch 440Hz. The way to synthesize a vibrato tone is explained in section 4. Figure 2 shows its waveform and its frequency domain information using FFT.

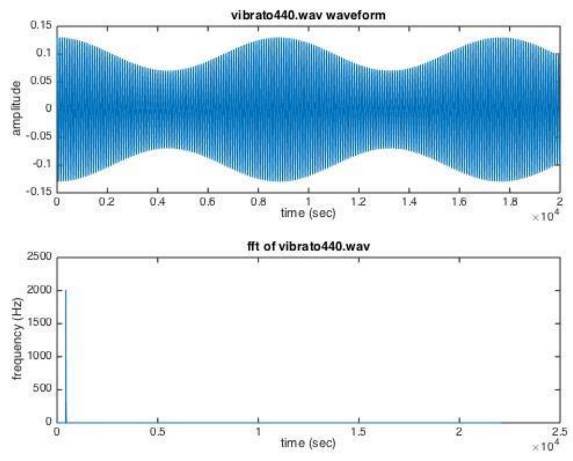

Figure 2

In Figure 3, we perform a short time Fourier transform on this wave form, But we can see the frequency over time is a flat bar. But actually, its frequency is supposed to be modulated. We can see it by observing the waveform in Figure 1. So this means we need more accurate algorithm to detect the FM. In Figure 4 and Figure 5, each shows the FM and AM detection results using our algorithm. We can clearly see the frequency is modulated around 440 Hz, from 443Hz to 437Hz. But as you can note from Figure 5, there are some error on the detected frequency depth. The peak of the sinusoidal wave are distorted, which should be perfect sine curve in our synthesized tone.

Method 2:
The idea of this method is to find the optimized peak instead of the peak bin we got from our FFT spectrum. Given 2 adjacent bins of the peak, we have three bins: X_{k-1}, X_k, X_{k+1}. The non-integer optimized bin number from the bin k is

$$\delta = \frac{|X_{k+1}| - |X_{k-1}|}{4|X_k| - 2|X_{k+1}| - 2|X_{k-1}|}$$

But based on the paper [4] (Jacobsen & Kootsookos, 2007), this frequency estimator gives a poor result on los signal-to-noise ratio condition. In our real time situation, we need our algorithm to be robust. So we follow the improved method in that paper. Instead of using magnitudes, we use the complex FFT values.

$$\delta = -Re[\frac{(X_{k+1}) - (X_{k-1})}{(2X_k - X_{k+1} - X_{k-1})}]$$

The optimized peak is $k + \delta$. Figure 6 shows the FM depth result of the same signal in Figure 2. We can see from this figure, it is accurately enough to show us the instantaneous frequency. A surprise that this algorithm gives us is it does not have the distorted portion on the peak part of the sinusoidal wave.

We are still testing both of the algorithms with more test tones also we are trying to code these algorithm in the real time tool. But in our real time system, we do now need to perform a short time Fourier transform in the first place, since every time chunk we take will be one frame in the case of STFT. More detail about this will be discussed in the later sections.

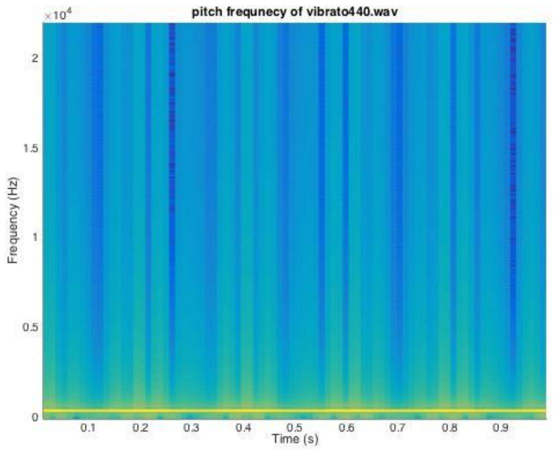

Figure 3

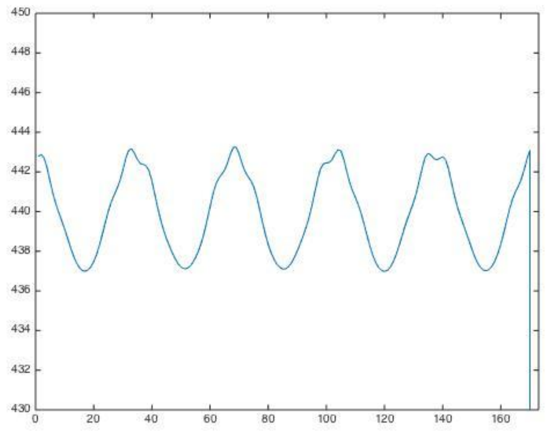

Figure 4

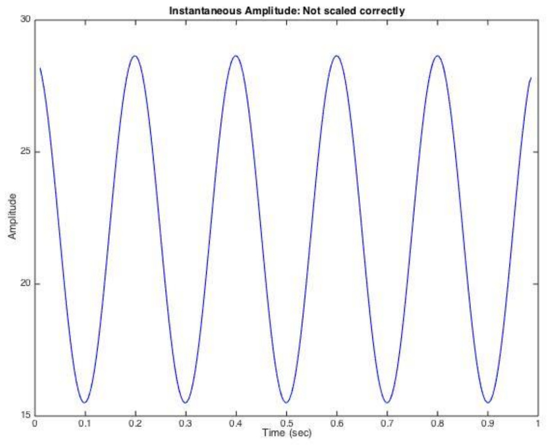

Figure 5

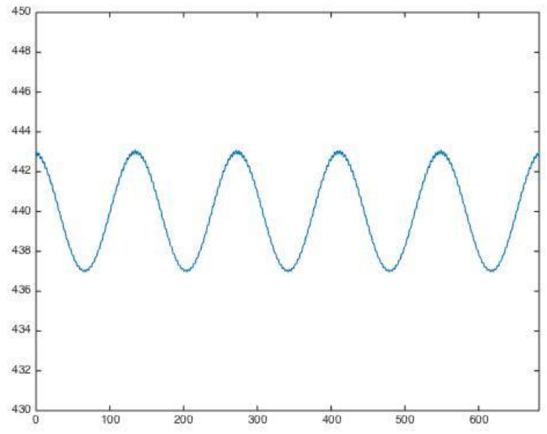

Figure 6

Step 4: vibrato rate estimation:
After we have detected the FM depth series, we take the results of several frames in the signal as our new "time-domain" signal. For example, in Figure 4, we have our FM depth vector and each data in this vector is calculated using a frame in our signal in actual time domain, in Figure 2. We perform the same technique on our FM depth signal. We choose a frame length of 128 and apply FFT. So the peak we get from each of the frame will be our frequency modulation rate. Since the FM rate does not change much, so a hop size of 64 should be good to estimate it.
But this step is not successfully working yet. The second FFT give us a bad result on low frequency detecting as the vibrato rate is usually 3Hz to 10Hz. We will continue working on this part.

4. TESTING TONE MAKING

In order to test our algorithm, we need to know the ground truth of the parameter of a vibrato. One way to get the ground truth is to test a pre-recoded tone using an accurate pitch detecting algorithm, like YIN. But these algorithms only serve for FM depth detection. We need to compare the results of the vibrato rate also. So this situation shows the imperative of making test tones. A test tone is a synthetic signal with known pitch, vibrato depth (AM, FM), and vibrato rate. The rest part of this section illustrate how to exactly make a synthetic vibrato tone.
AM simulation:
Parameters: initial AM phase: ϕ_{ai}. AM rate: f_a. Time: t. AM depth: A_a.
So we apply these parameters into a sinusoidal wave, which then becomes our amplitude modulation:

$$AM = 1 + A_a \cos(2\pi \times f_a \times t + \phi_{ai})$$

FM simulation:
Parameters: initial FM phase: ϕ_{fi}. FM phase: ϕ_t FM rate: f_f. Time: t. FM depth: A_f. Fundamental frequency f_0.
Like AM, we first build our frequency modulation source with its rate.

$$\Delta f = \cos(2\pi \times f_f \times t + \phi_{fi})$$

Not compared to AM, Δf is in unit of Hz. But Δf is not our final frequency modulation. We need to scale Δf with our FM depth factor. Also we need to record the instantaneous phase increment based on Δf.

$$\phi_t = \phi_{t-1} + 2\pi \times A_f \times \cos(2\pi \times f_f \times t + \phi_{fi})$$

So we will have a vector of phase information ϕ_t depends on time. Having the phase information, we can build our simulated vibrato signal as
$vib(t) = (1 + A_a \cos(2\pi t f_a + \phi_{ai}) \times \cos(2\pi t f_0 + \phi_t)$.
Since vibrato depth tends to increase on higher partial we need to adjust the FM and AM depth when synthesizing a note with multiple harmonics (Maher & Beauchamp, 1990).

5. SOFTWARE STRUCTURE

In this section, we will briefly introduce the overall structure of the real time software that visualizes the music vibrato. It contains 3 big unit, a real time audio data I/O, a DSP processing unit and a graphic visualizing unit. The whole program is built under Mac system using C/C++.

4.1. Real Time Data I/O
This part's functionality is to take a fixed time length data from microphone and store it in a buffer, read to be processed by the DSP unit. The C++ library we used to code this unit is PortAudio. It is a very popular audio API in C language. The reason we chose this library is: it's a cross platform library, which means it works on all the systems, like Windows, Mac, Linux, etc. But in our project, we only programed it in Mac system. And in this case PortAudio acts like an intermedia proxy which help communicate between the program we wrote and the apple embedded low level audio API CoreAudio. Another reason is this API actually can perform sample by sample processing, if needed. So this really gives us so much freedom to our design. But due to our algorithm, we cannot make each buffer too small by losing the accuracy of the detection. But this API gives us the potential to make our algorithm be better.

4.2. Real Time DSP Unit
In this unit, we perform our core algorithm, the time frequency processing here. Instead of an offline signal, the data gathered by PortAudio will periodically update, the period is the hope size of our frames in time. So each period gives us the time to do the processing job. In our program, we set the length of each frame to be 1024 samples, if the sampling frequency is 44.1kHz, then the time length of each frame will around 23ms. This also tells us the time latency. So when the buffer is filled up, the DSP will perform the algorithm in section 3 then puts the result in another shared memory, either an allocated memory or static memory. This memory will then be ready to bed used by the graphic unit.

4.3. Graphic Visualization Unit
The graphic visualization unit is programmed with the help of OpenGL. OpenGL is also a popular computer graphic library used in many fields like animation, game programming and so on. The reason we choose to use this harder programed library rather than some math graph library is our ultimate goal to make it a game like scene. We hope we can use the vibrato that detected by the algorithm to control some kind of character in the screen. And OpenGL has no trouble doing any of that. Currently we can only visualize the real time spectrum of the each frame. Figure 5 shows a screenshot of our visualization.

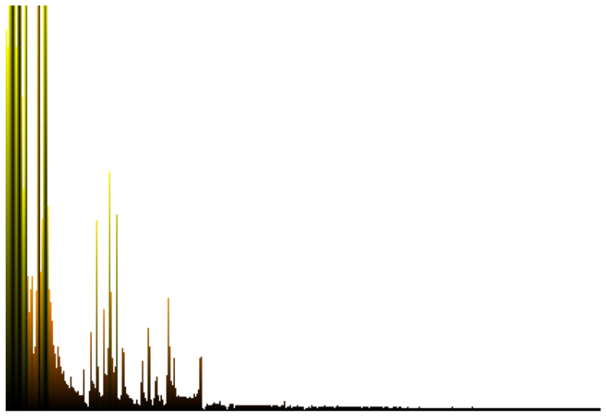

Figure 7

In this screen shot we can see the color change depending on the height of each peak. We meant to program it that way so it looks like a fire. So the freedom on color, 3D graphic, texture and so one are the feature that other waveform oriented graphing tools do not have.

Combining the OpenGL and Portaudio is not a trivial task. Besides thoroughly reading the reference of the functions in both APIs, we also need to perform a multi threads programing technique. Since PortAudio real time I/O and OpenGL rendering use infinity loop to do their jobs. We created a new thread on PortAudio so that they will not interrupt each other.

Since we cannot show a picture of the ultimate visualization, we will just discuss a realizable simple demonstration in word. After the graphic unit gets the vibrato data, we will try draw a 3D plot on FM vs AM vs time. Time will our z axis which goes into the screen. A dot on the screen will show how much FM and AM your note has at this moment. With time going up, the dots will be driven by your vibrato, which makes this basic role play game like visualization.

6. FUTURE WORK

The main body of this paper already mentioned some future work we need to. The first one will be to finish up the real time vibrato detection implementation. Secondly, we need to add some filter library to help clean the note if from a noisy environment. Third, we hope to have some better graphic character, instead of a dot, maybe we can make a plane, which looks more reasonable for a user to control.

We also know there are many bugs exited in the program in current phase. We will keep debugging on it. We hope to create some novel musical game in the end.

7. REFERENCES

[1] V. Verfaille, C. Guastavino, and P. Depalle, "Perceptual evaluation of Vibrato Models," *CIM05*, Publisher, Montreal, pp. 1-10, 2005.

[2] Udo Z., *DAFX: Digital Audio Effect*, John Wiley & Sons, Ltd, , 2011.

[3] R. Mahar, J. Beauchamp, "An Investigation of Vocal Vibrato for Synthesis," *Applied Acoustics* 0003-628X, 1990

[4] E. Jacobsen, P. Kootsookos, "Fast, Accurate Frequency Estimatros," *IEEE Signal Projcessing [125]*, May, 2007