Rock Roll and the Gabor Transform

Haoran Sun

February 6, 2021

Abstract

The Fourier Transform is very useful under many situations. However, a trade-off between the time and frequency information will occur. For being able to obtain both time and frequency information, this text introduces the method of the Gabor Transform,which analyzes a subdomain of the signal at each time and combine all the subdomains information.In practice, We use this method to analyze the portions of two songs' clips, finding out which notes are being played at certain time.

1 Introduction and Overview

This text describes an application of the Gabor Transform in practice: The clips of two greatest rock songs *Sweet Child O' Mine* by Guns N' Roses and *Comfortably Numb* by Pink Floyd are provided, and we will reproduce the music score(by finding what notes are being played) of the guitar in the first song, and the bass in the second utilizing the Gabor Transform. We then will isolate the bass in *Comfortably Numb* by applying a filter centered at bass notes' and attenuating the overtones.

1.1 Notes

Every musical notes have their corresponding frequencies. The higher the frequency is, the higher the pitch of a note is (See Figure 1 for the frequency in Hz each note has). Therefore, we can know what notes are being played by looking at the frequencies shown in the signal.

1.2 Filtering

Once we decide the frequencies of bass notes in the Floyd clip, we will be applying a filter to the signal in its frequency domain, which is able to eliminate overtones and notes of other instruments, while maintain the bass notes. In this task, we will simply use Gaussian function as our filter. In the same way, we can use the filter to try to extract information of the guitar solo in the Floyd clip.

2 Theoretical Background

2.1 Gabor Transform

Different but very similar to the Fourier Transform, we define the Gabor Transform(also named as STFT, the short-time Fourier Transform) of a given function $f(t)$, where $x \in \mathbb{R}$, written as $\tilde{f}_g(\tau, k)$, by the formula:

$$\tilde{f}_g(\tau, k) = \int_{-\infty}^{\infty} f(t)g(t-\tau)e^{-ikt}dt \tag{1}$$

This equation decomposes the given function, which is in spatial or time domain, into function depending on frequency(as what the Fourier Transform does). However, it only gives you the information in frequency domain near time τ. The function $g(t-\tau)$ acts like a filter(but in time rather than in frequency domain) and provides the information of only a subdomain of the whole signal. Taking the Fourier Transform of the subdomain helps isolate the frequencies that present in the subdomain. By combining the transform of

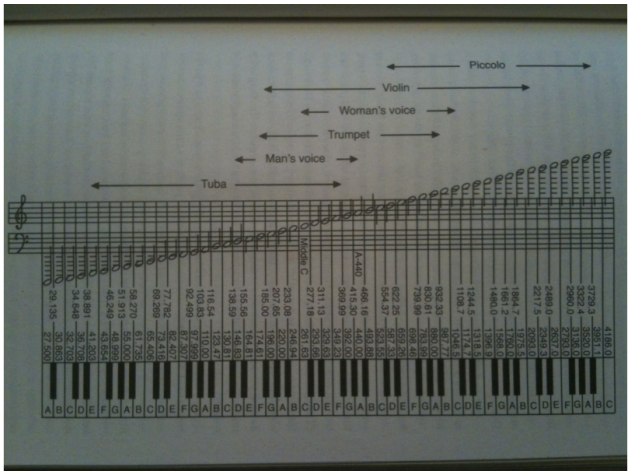


Figure 1: Music scale along with the frequency of each note in Hz

different subdomains, we will understand when each frequency is present in the whole signal. The function $g(t-\tau)$ must satisfy the following conditions:

1. The function g must be real and symmetric;
2. The L_2-norm of g is set to unity.

The simple Gaussian filter satisfies both conditions, and we will be using it in our tasks.

2.2 Discrete Gabor Transform

Similar to the Fourier Transform, in practice, we always can only obtain limited discrete data points and continuous Gabor Transform is not applicable. Therefore, a discrete version of the Gabor Transform is needed. We define the Discrete Gabor Transform as:

$$\tilde{f}_g(m,n) = \int_{-\infty}^{\infty} f(t)g(t-nt_0)e^{2\pi i m\omega_0 t}dt \tag{2}$$

where $k = m\omega_0; \tau = nt_0$. Both m and n are integers. ω_0 determines the size of the window, and t_0 determines the time step between subdomains. We want a small ω_0 in order to have good frequency resolution, but a large enough window to have overlaps between subdomains so that we don't lose information at the boundaries.

3 Algorithm Implementation and Development

The audio clip of the song will be read as a vector representing the music. It contains the information of the audio signal in time domain. We will be analyzing and editing the clips by modifying the vector. We set our spatial domain to be the length of the audio clip in seconds. And the function will be sampled at equally-spaced n points, where n to be the length of the audio vector. Instead of rescaling the frequency domain by $2\pi/L$, we will be using $1/L$ as the coefficient(since the frequencies of notes are read in Hertz).

3.1 Visualization in time-frequency domain

In order to visualize both time and frequency information, we will apply the Gabor Transform to the audio signal and plot the spectrogram,which contains time domain on the horizontal direction and frequency on the vertical direction(see Algorithm **??**), we will:

1. Read the audio clip into the audio vector and set up a time vector with a decided time-step, which is 0.1 in this task
2. Create a loop to go over the data at different time.
3. Inside the loop, apply the Gabor filter(a = 100) centered at the corresponding time to the signal and transfer it into frequency domain.
4. Store the frequency information into a matrix, where each column will be matched with a time value in the spectrogram and each row will be matched with a frequency value.
5. After the loop, we plot the matrix using the spectrogram and analyze where the notes are being played.

Algorithm 1: Spotting notes

```
Import data from the audio clip into the vector y
Build a vector containing time info with steps named sum
for j = 1 : length(sum) do
    Define the Gabor Filter g
    Apply g to y by multiplying them element-wisely.
    Take FFT of the filtered vector using fft()
    Record the transferred data into matrix on column(j)
end for
Plot the frequency signal in time-frequency domain using spectrogram
```

3.2 Isolating Bass in Floyd Clip

Once we obtain the spectrogram for the the Floyd clip in section 3.1, we are able to determine what the bass notes' frequencies are and locate our filter centered at the note's frequency at each corresponding time in order to filter out overtones and notes played by other instruments. The main task is to come up with an algorithm(See algorithm 2) that can update the center of the filter as the time changes, since the notes being played are changing. To accomplish this, we will:

1. Follow the same steps as Step 1 to 4 from 3.1
2. Find the position of any frequency whose value is below 150Hz(the typical frequency of a bass note) in the frequency vector
3. Find the corresponding signal amplitude corresponding to those frequencies.
4. Compare their amplitudes, and take the frequency that has the maximum signal amplitude to be our filter center(we use Gaussian for this frequency filter as well).
5. Because the signal is symmetric about 0, we will have two frequencies in step 4 (positive and negative). So we actually have two filters with different centers symmetric about 0
6. Apply these two filters to the signal separately in frequency domain, add them up, store it in its corresponding position in the matrix, and plot it using spectrogram.
7. We can take the inverse fourier transform of the filtered signal each time the loop runs and add them into a vector, which will be the vector of a filtered audio signal with bassline only.

Algorithm 2: Filtering

Import data from the audio clip into the vector `y`
Build a vector containing time info with steps named `sum`
for $j = 1 :$ `length(sum)` **do**
  Define the Gabor Filter `g`
  Apply `g` to `y` by multiplying them element-wisely.
  Take FFT of the filtered vector using `fft()`
  Find the elements whose value is less than 150 in the frequency vector `ks()` and find the corresponding signal amplitude
  Get the maximum amplitude using `max()`, then find the index of this amplitude in the signal amplitude vector using `find(vec = Maximum)`
  Find the element at this index inside frequency domain, use this frequency value as the center of the filter
  Apply the filter to the data and record the transferred data into matrix on column(j)
  apply the inverse Fourier Transform using `ifft()` to the filtered signal and add them into a vector
end for

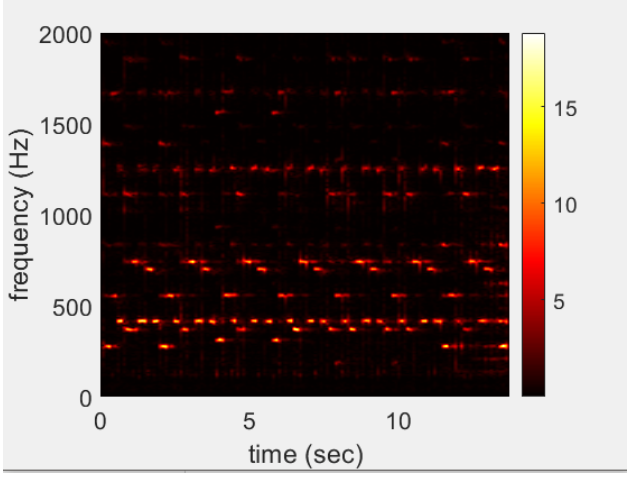


Figure 2: Spectrogram of the whole GNR Clip

4 Computational Results

4.1 Identifying the notes

4.1.1 Guitar in GNR Clip

By executing Algorithm 1, we successfully obtained the spectrogram of the GNR clip in Figure2. Since the frequency are symmetric about 0, we can only look at the positive value of the frequencies. We set the range of frequency in the graph to be from 0 to 2000Hz. It is clearly identifiable there is a pattern of the brightest spots in the spectrogram approximately between 250Hz and 1000Hz, which is reasonable, since the guitar riff repeats a lot, and notes between each bars are very similar except few of them are off. With this being said, we zoom into only the first bar of the clip($\sim$ first two seconds), shown in Figure 3. The Y value of each data tips show the value of the frequencies of each note(I have to stretch the picture horizontally so that data tips will not block each other). Referring to Figure 1, we can tell the notes are being played in the first bar are $[C_4^{\#}, C_5^{\#}, G_4^{\#}, F_4^{\#}, F_5^{\#}, G_4^{\#}, F_5, G_4^{\#}]$. The very similar result are obtained for both second bar and third bars. The only difference between these three bars is their first note, in which the second bar starts with $D_4^{\#}$(f = 310.5Hz in the spectrogram) and the third bar starts with $F_4^{\#}$(f = 370.4Hz in the spectrogram). Each bar repeats twice.

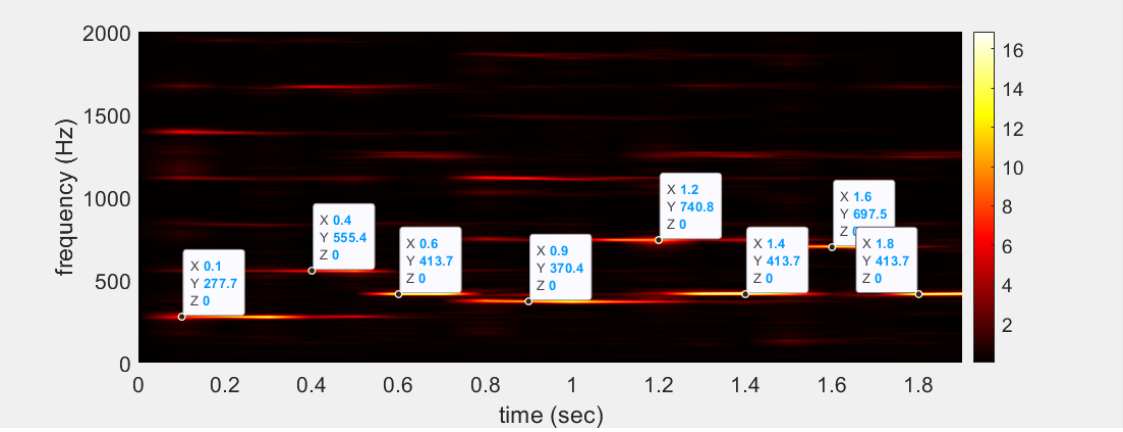


Figure 3: Spectrogram of the first bar of GNR clip

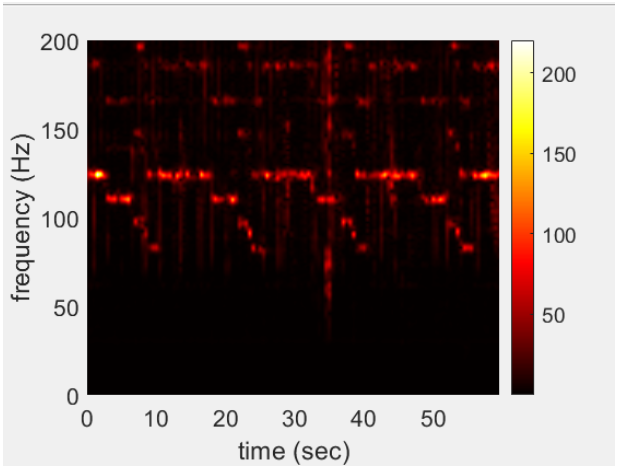


Figure 4: Spectrogram of the whole Floyd Clip under 200 Hz(Left)

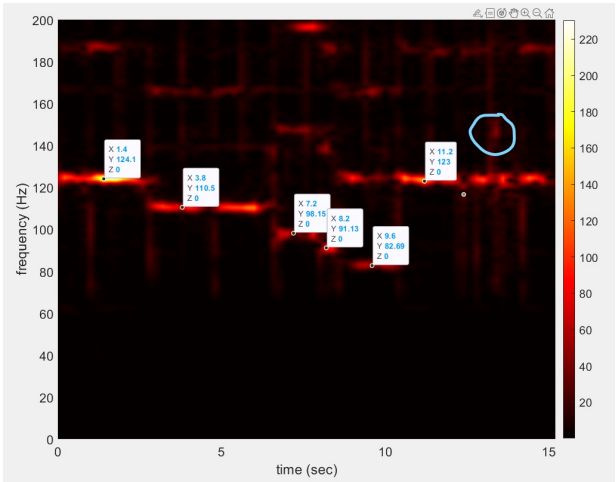


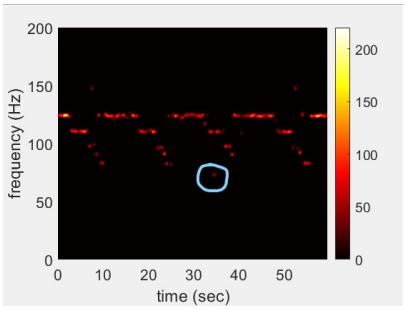
Figure 5: Spectrogram of the first section under 200 Hz(Right)

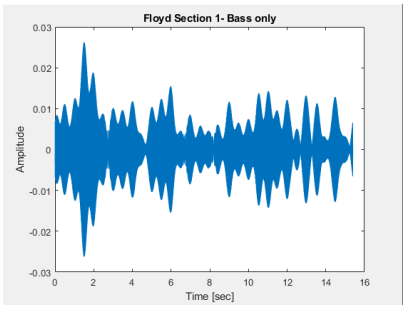
4.1.2 Bass in Floyd Clip

As in section 4.1.1, we execute Algorithm 1 and obtained the spectrogram of the Floyd clip (see Figure 4). Different from the GNR clip, since we only need to look at the bass notes(which typically have low frequencies), the frequency range of the spectrogram is set to be from 0 to 200. As the GNR clip, a clear pattern appears. We zoom in to the first section (first 15 sec of the the clip, see Figure 5). Again, we refer to Figure 1, the bass notes being played in the section are $[B_2, A_2, G_2, F_2^{\#}, E_2, B_2]$. One thing to notice is, there is a little bright spot shown in Figure 5(the blue circled region) at frequency 146.3Hz, corresponding to note D_3. It is not likely to be an overtone, otherwise we should be able to see the spot of the note D_2 at the same time. With this being said, D_3 is probably a very short note played by the bass as well.

4.2 Isolating Bass in Floyd Clip

Using Algorithm 2 introduced in Section 3.2, we obtained Figure 6 as the filtered spectrogram for bass. The result is satisfying. Most of the bass notes are maintained while the overtones are filtered out except a few spots of noise. The reason is other low-pitch instruments overtake the bass at certain time. For instance, there is a tiny fluctuation in the blue circled region in Figure 6, where the bass drum were played.





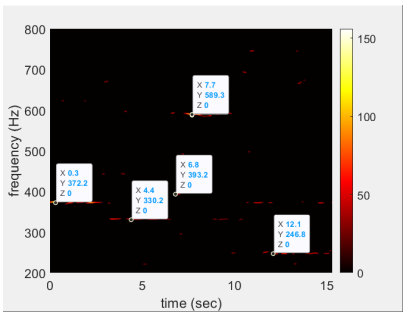


Figure 6: Spectrogram of the filtered whole Floyd Clip under 200 Hz(Left)

Figure 7: Audio visualization of bassline of Floyd Clip(Middle)

Figure 8: Spectrogram of the filtered first section from 200Hz to 800Hz (Right)

The visualization of the audio clip with only bass part of the first section of Floyd clip(first 15 sec) is shown in Figure 7. However, it is not playable in Matlab, and therefore this result should be taken with a grain of salt.

4.3 Isolating Guitar Solo in Floyd Clip

For find the notes of the guitar solo, we apply the same implementation as Section 4.2. Instead of looking at the frequency below 150Hz, we take 200Hz to 800Hz to be our frequency range, since guitar notes have a much higher frequency than bass notes. The result are shown in Figure 8. Because the quick variation of the guitar notes and mix with other instrument, it is difficult to spot every single notes of the guitar solo. Therefore, we will be only looking at the bright and sustaining signal in the figure. Again, comparing the frequencies with the music scale in Figure 1, the notes we can identify include:$[F_4^{\#}, E_4, G_4, D_5, B_3]$

5 Summary and Conclusions

In short, this text introduces the method named Gabor Transform, which maintains both time and frequency information of the signal. We also see how this method can be applied in practice by spotting which notes are played at certain time in audio clips.

Since the guitar notes in GNR and the bass notes in Floyd clips are clearly identifiable and the music score of the bass notes is clean after filtering, the overall result we achieved meets expectation decently. We obtained a visualization of the bass-only audio clip for part of the Floyd song, but the correctness is unknown(very likely wrong) since we failed to play it. We only identified 5 notes for the guitar solo in first section of the Floyd clip because of the quick variation of the notes and mixed-up instrumental lines. Also, we only looked at a portion of the Floyd clip for the guitar spectrogram, since it takes forever for Matlab generating that for the whole clip.

Appendix A MATLAB Functions

- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- `M = max(X)` returns the largest element of vector `X`.
- `fft(X)` returns discrete Fourier transform of vector X.
- `fftshift` shifts zero-frequency component to center of spectrogram.
- `pcolor` is a Pseudocolor (checkerboard) plot, a flat surface plot viewed from above.

Appendix B MATLAB Code

```
clear; close all; clc


[y, Fs] = audioread('GNR-third bar.m4a');
L = length(y)/Fs; % record time in seconds
n = length(y);
t2 = linspace(0,L,n+1); t = t2(1:n);
k = (1/L)*[0:n/2-1 -n/2:-1];
ks = fftshift(k);%frequency domain

a = 1000;
tau = (0:0.1:L);

%% Gabor Transform and Spectrum
for j = 1:length(tau)
    g = exp(-a*(t-tau(j)).^2); %window function
    yg = g.*y';
    ygt = fft(yg);
    ygt_spec(:,j) = fftshift(abs(ygt));
end

figure(1)
a = pcolor(tau,ks,ygt_spec);
shading interp
set(gca,'ylim',[0 2000],'Fontsize',16)
colormap(hot)
colorbar
xlabel('time (sec)'), ylabel('frequency (Hz)')
```

Listing 1: Code for plotting spectrogram of audio clips

```
clear; close all; clc


[y, Fs] = audioread('Floyd-section1.m4a');
L = length(y)/Fs; % record time in seconds
n = length(y);
t2 = linspace(0,L,n+1); t = t2(1:n);
k = (1/L)*[0:n/2-1 -n/2:-1];
ks = fftshift(k);%frequency domain

a = 50;
tau = (0:0.5:L);
ygt_spec = zeros(length(y),length(tau));

%% Gabor Transform and filtered Spectrum
yf = zeros(1,length(y));
for j = 1:length(tau)
    g = exp(-a*(t-tau(j)).^2); %window function
    yg = g.*y';
    ygt = fft(yg);
    ygt_spec(:,j) = fftshift(abs(ygt));
    temp = ygt_spec(:,j);

    %looking for the bass frequency(<150Hz) for centering the filter.
    %looking for the maximum value of ygt_spec of any frequency below 150 Hz
    ks_abs = abs(ks);
    ks_bass_position = ks_abs < 150; %position of bass notes frequency in the frequency vector
    ks_bass = temp(ks_bass_position);
    M = max(ks_bass);
    I = find (temp == M);
    center1 = ks(I(1));%central freuqncy of the filter
    center2 = ks(I(2));
    %Define the frequency filter
    b = 0.3;
    filter1 = exp(-b*(ks-center1).^2);
    filter2 = exp(-b*(ks-center2).^2);
    ygt_spec(1:length(ks),j) = temp(1:length(ks)).'.*filter1+temp(1:length(ks)).'.*filter2;

    %Transfer the filtered signal back to "music" domain
    ygt = fftshift(ygt);
    ygtf = ygt.*filter1+ygt.*filter2;
    ygf = ifft(ygtf);
    yf_time = ygf;
    yf = yf+yf_time;
end
plot((1:length(y))/Fs,yf);
xlabel('Time [sec]'); ylabel('Amplitude');
title('Floyd Section 1- Bass only');
%Spectrogram of the isolated bass notes
figure(1)
a = pcolor(tau,ks,ygt_spec(1:length(ks),:));
shading interp
set(gca,'ylim',[0 200],'Fontsize',16)
colormap(hot)
colorbar
xlabel('time (sec)'), ylabel('frequency (Hz)')
```


Listing 2: Code for isolating the bass in Floyd clip

```
clear; close all; clc


[y, Fs] = audioread('Floyd-section1.m4a');
L = length(y)/Fs; % record time in seconds
n = length(y);
t2 = linspace(0,L,n+1); t = t2(1:n);
k = (1/L)*[0:n/2-1 -n/2:-1];
ks = fftshift(k);%frequency domain

a = 60;
tau = (0:0.1:L);
ygt_spec = zeros(length(y),length(tau));

%% Gabor Transform and filtered Spectrum
yf = zeros(1,length(y));
for j = 1:length(tau)
    g = exp(-a*(t-tau(j)).^2); %window function
    yg = g.*y';
    ygt = fft(yg);
    ygt_spec(:,j) = fftshift(abs(ygt));
    temp = ygt_spec(:,j);

    %looking for the bass frequency(>200Hz) for centering the filter.
    %looking for the maximum value of ygt_spec of any frequency over 200 Hz
    ks_abs = abs(ks);
    ks_bass_position = ks_abs > 200; %position of bass notes frequency in the frequency vector
    ks_bass = temp(ks_bass_position);
    M = max(ks_bass);
    I = find (temp == M);
    center1 = ks(I(1));%central freuqncy of the filter
    center2 = ks(I(2));
    %Define the frequency filter
    b = 0.3;
    filter1 = exp(-b*(ks-center1).^2);
    filter2 = exp(-b*(ks-center2).^2);
    ygt_spec(1:length(ks),j) = temp(1:length(ks)).'.*filter1+temp(1:length(ks)).'.*filter2;
end

figure(1)
a = pcolor(tau,ks,ygt_spec(1:length(ks),:));
shading interp
set(gca,'ylim',[200 800],'Fontsize',16)
colormap(hot)
colorbar
xlabel('time (sec)'), ylabel('frequency (Hz)')
```

Listing 3: Code for isolating the guitar solo in Floyd clip