REAL-TIME MUSICAL ANALYSIS OF POLYPHONIC GUITAR AUDIO

A Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

John Hartquist

June 2012

COMMITTEE MEMBERSHIP

TITLE:	Real-time Musical Analysis of Polyphonic Guitar Audio
AUTHOR:	John Hartquist
DATE SUBMITTED:	June 2012
COMMITTEE CHAIR:	John Clements, Ph.D.
COMMITTEE MEMBER:	Zoë Wood, Ph.D.
COMMITTEE MEMBER:	Franz Kurfess, Ph.D.

Abstract

Real-time Musical Analysis of Polyphonic Guitar Audio

John Hartquist

In this thesis, we analyze the audio signal of a guitar to extract musical data in real-time. Specifically, the pitch and octave of notes and chords are displayed over time. Previous work has shown that non-negative matrix factorization is an effective method for classifying the pitches of simultaneous notes. We explore the effect of window size, hop length, and other parameters to maximize the resolution and accuracy of the output.

Other groups have required prerecorded note samples to build a library of note templates to search for. We automate this step and compute the library at run-time, tuning it specifically for the input guitar. The program we present generates a musical visualization of the results in addition to suggestions for fingerings of chords in the form of a fretboard display and tablature notation. This program is built as an applet and is accessible from the web browser.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Music is all around us. We hear it on the radio, on television, in movies, at coffee shops, and even on our cell phones. As general listeners, we do not need to have a deep understanding of the science behind music to have an appreciation for it. Musicians, on the other hand, often spend years learning the intricacies of their art [23]. They dig deep into the theory, and practice routinely to perfect their technique. Over time, they are able to train their ears to hear tiny differences in sound that an average person would barely be able to perceive.

With the rise of computers over the past few decades, many tools have been created to aid musicians in a variety of ways. Digital tuners are common, and software for recording and editing audio is readily available [23]. One area that is still being actively researched is that of music analysis. Computer programs have been written to try to identify the pitch and duration of specific notes in audio files for the purpose of automatic music transcription [28]. Many solutions have been proposed, but most of them are limited in their accuracy, and most are geared for offline use. That is, they work well for prerecorded audio tracks, but are not designed for real-time audio input. Other limitations include the inability to distinguish between individual instruments when there are many present. While different instruments create dramatically different types of sounds, even a single instrument can have a very wide range of sounds. While it is easy for humans to recognize sounds as belonging to a particular instrument, it is a difficult task for computers.

In this thesis, we focus on the audio signal of a single guitar, and aim to analyze and visualize musical information in real-time. In particular, we attempt to locate notes as they are played, and classify the exact pitch and duration of each one. This thesis works toward finding an efficient method that can later be used to aid musicians in practice and performance, giving them a visual interpretation of the sound they are creating, as well as providing musical analysis of the notes that they are playing.

The guitar is a dynamic instrument, and many guitar players express notes in different ways. Sometimes they slide from one note to another, and sometimes they manipulate the pitch of notes by bending them in various ways. Notes can be played at different volume levels, as well as with a variety of effects such as distortion or reverb. Despite these challenges, we desire high resolution in both the time and frequency domains.

Music can be classified into two main categories, monophony and polyphony. We call music monophony when there is only one note occurring at any given time. When there are multiple notes at a time, it can be called polyphony. While there are many techniques to detect the pitch of a single note, detecting many simultaneous notes can be much more difficult. On the guitar, there can be as many as six simultaneous notes (one on each string) that when played together are called chords.

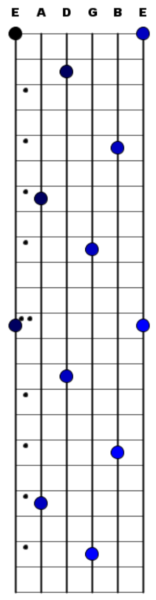


Figure 1.1: All E notes on the guitar neck. Different colors represent different octaves.

To further complicate things, notes with the same pitch can be played from different strings. For example, the note B_2 (with a fundamental frequency of 123.46 Hz) may be played by plucking the sixth string at the 7th fret or by playing the fifth string at the 2nd fret. Some notes have as many as five different locations on the fretboard. Due to this, chords of up to six notes can often be played in different positions along the guitar neck.

We attempt to accurately display these notes in an informative manner as they are being played. To do this we build from existing work related to an algorithm utilizing non-negative matrix factorization [3, 8, 12, 17]. Specifically, we eliminate the requirement for prerecorded note samples, explore different window sizes, and utilize multi-resolution FFT analysis to efficiently determine the pitch and octaves of simultaneous sounding notes. In addition, musical analysis of the notes relating to harmony and melody is provided in real-time, as well as possible hand positions and fingerings for chords.

Finally, this thesis attempts to make such a tool as available as possible to musicians by requiring minimal steps to get the software running. It is developed as a Processing (www.processing.org) applet that runs inside the browser, and is accessible by simply visiting a website. As of now, no other such tool is known to exist.

Chapter 2

Background

2.1 Music Theory

Before we discuss the goals of this project, it is important to understand some basic concepts of how music works. Music is made up of individual notes that each have characteristics such as pitch, duration, loudness, and timbre.

Pitch is an audible perception of sound in relation to how humans hear. It is how we perceive tones to be higher or lower when compared to one another. Western music characterizes pitch into 12 different semi-tones which when grouped together are called an **octave**, with octaves being repeated over and over. A **pitch class** refers to one of the tones, for example the note B, and includes all instances of B across all octaves. While pitch is logarithmically related to the frequency of the waveform making up an audio signal, technically it is just describing how humans perceive music [28].

Duration quantifies how long a certain note lasts, and is measured in "beats" or fractions of beats which are in turn grouped into measures.

Timbre is often referred to as the “color” of the note. It is what allows humans to hear the difference between a guitar and a piano, even if they are played at the same pitch. In addition, similar instruments may have different timbres. For example, two different models of an electric guitar may sound slightly different. While timbre is easy for humans to distinguish, it is a lot harder to write a computer program to do the same thing.

Often times, notes are described in the context of scales. If 12 consecutively pitched notes are grouped together and played in succession, they are called a chromatic **scale**. Other scales, made up of fewer notes can be constructed, and characterized by the intervals between each note. Major scales, are considered to sound “happy”, and minor scales are considered to sound “sad”.

Chords occur when more than one note is played at the same time. They are named according to which notes from the scale are present. For example, if the chord includes the first, third, and fifth notes of the major scale, it’s considered to be a major chord. There are many different types and variations of chords.

On many instruments, it is possible to play the same note in different ways. For example, on a guitar, the same note may exist in various places on different strings. To complicate manners, guitarists use many different techniques when playing notes. Some use their fingers to pick notes, and some people use a plectrum (also called a pick). They often use different volumes, and sometimes alter notes by bending the strings or moving quickly from one note to another. In this thesis, we strive to gather enough data to analyze the expression of each note as much as possible in addition to identifying its pitch.

Octave	Pitch
2	. . .
	A
	A$^{\#}$/Bb
	B
3	C
	C$^{\#}$/Db
	D
	D$^{\#}$/Eb
	E
	F
	F$^{\#}$/Gb
	G
	G$^{\#}$/Ab
	A
	A$^{\#}$/Bb
	B
4	C
	C$^{\#}$/Db
	D
	. . .

(a) Chromatic Scale

Octave	Pitch
2	. . .
	A
	B
3	**C**
	D
	E
	F
	G
	A
	B
4	**C**
	D
	. . .

(b) C Major scale

Octave	Pitch
2	. . .
	B
3	**C**
	E
	G
	B
4	**C**
	. . .

(c) C Major7 Chord

Figure 2.1: (a) All the pitch classes form a chromatic scale (b) Only the notes included in the C major scale (c) The notes that form a C major 7 chord

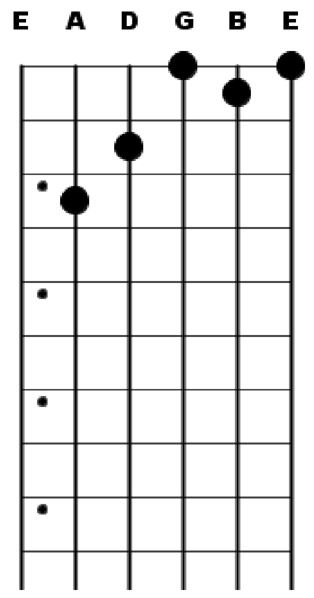


Figure 2.2: Fretboard displaying an open C major chord

2.1.1 Guitar

This thesis focuses on analyzing the sound generated by electric and acoustic guitars. Standard guitars have six strings and usually at least 21 frets. Strings are numbered from the right to the left when looking directly at the fretboard, with the first string having the highest pitch and the sixth string having the lowest pitch. Individual notes are played by pressing down on a string above a certain fret and picking (plucking) the string. As the frets get closer to the body of the guitar, the pitch of its corresponding note increases in semi-tone intervals. Fingering charts can be used to describe which frets must be held down on different strings for a specific chord to sound. Figure 2.2 shows how to play an open C major chord.

One of the characteristics of the guitar is that there are many different ways to play the same note. In standard tuning, the lowest note of the guitar is the E on the open sixth string. Five frets up is an A note which sounds the same as the open note on the fifth string. These notes that have the same pitch but different locations are said to be enharmonic. Some notes may occur as many as five times throughout the neck. The notes of the open strings from the sixth string to the first string in standard tuning are EADGBe with the capital E being two octaves lower than the lowercase e. In addition to standard tuning, guitarists may prefer to use an alternate tuning, where strings are tuned different to notes, for example DADGAD. This changes the location of notes throughout the guitar neck, thereby creating different hand and finger positions for making chords.

To actually produce the sound, acoustic guitars have a hollow body where the vibrations from the strings resonate. In contrast, electric guitars use magnetic coils at the bridge of the guitar to translate the vibrations into an electrical signal that is fed to an amplifier. Some acoustic guitars also have a microphone preamp built into them, allowing them to connect to an amplifier as well.

Throughout this thesis, we use an electric and acoustic guitar with analog outputs that connect to a Line 6 GuitarPort, an analog-to-digital converter, that connects to a computer via USB.

2.1.2 Harmonics

When we talk about the pitch of a note in terms of its frequency, what we are really referring to is its **fundamental frequency**. For example, the note A_2 has a fundamental frequency of 110 Hz. Real-world musical notes are made up of the fundamental frequency as well as a number of **harmonics** that occur at integer multiples of the fundamental, e.g. 220, 440, 880, 1760 Hz, etc. This makes analysis difficult, because the first harmonic of the note (in this case, found at the frequency 220 Hz) also represents the fundamental frequency of a note with the same pitch class that is one octave higher (A_3). Research has shown that the strength and characteristics of harmonics have a lot to do with the overall timbre of a note.

2.1.3 Music Representation

There are many different ways for musicians to represent music. The most common is in the form of sheet music using standard musical notation. On instruments where the same note might have multiple different locations, it is difficult to directly map notes to hand positions. Only highly trained musicians can sight read musical notation. Instead, many guitarists use tablature, another music representation that uses numbers to represent fretted notes on different strings.

Musical Notation

Music notation is used by a wide variety of musicians to represent the musical structure of songs. It includes information about the dynamics of the song, tempo, key and time signatures, as well as all of its notes and their respective pitches. The dynamics refer to how loud or quiet parts of the song are supposed to be played. Tempo is the speed at which the piece should be performed, usually measured in beats per minute (BPM). The key signature contains information about which key the song is in as well as indicating which notes should be played sharp or flat (a semi-tone higher or lower) by default. The time signature has to do with how the notes are counted and divided into measures, or sections of beats. This, along with the the actual notes themselves determine the rhythmical structure of the music. Each note contains both information about its duration

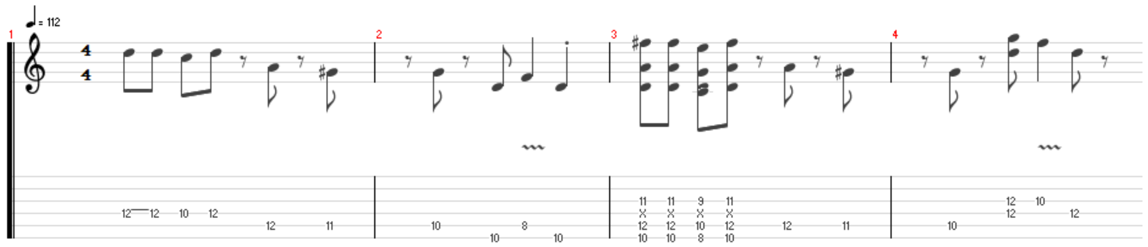


Figure 2.3: Music notation (top) and guitar tablature (bottom)

signified by the shape and if it is filled in or not, as well as its pitch information signified by where the note is located vertically on the staff. Rest notes may be marked as well, specifying the amount of time for silence between notes. Sheet music is read with time flowing left to right, with special markings for parts that might be repeated multiple times, or where to jump to after a chorus. See Figure 2.3 for a short piece of sheet music.

Guitar Tablature

An alternative form of music representation is called tablature, or tab for short. Specifically suited for fretted string instruments like guitar, it looks similar to music notation, but rather than signifying pitch, the horizontal lines each correspond to a single string of the instrument. Notes are designated by numbers that correspond to the fret that must be held down on each respective string. There are symbols to represent a variety of techniques and dynamics that can be traced loosely to standard music notation. While some tabs includes timing and rhythm information, the majority of tabs do not. There are many websites such as Ultimate-Guitar.com that host tabs for popular music [1].

These websites allow people to create their own tab and share them with the world. Creating tablature can be time consuming and is very error prone.

Because of this, many tablature online is of poor quality and is often based on guesswork. In many cases, not only are tabs incomplete versions of the song, but they include incorrect hand positions as well as incorrect or missing notes. There are also different meanings for accuracy. One one side of the spectrum it could mean that every note is written exactly as played by the original recording. One the other side, the tab may be simply recording the correct pitches, or a simplified way of playing the song. Even with an education of basic music concepts, creating accurate tablature requires excellent hearing to distinguish minor differences between notes and chords.

2.2 Computers and Sound

Sound is made up of energy in the form of waves. When it is converted from an analog to digital signal, it is discretized into a series of samples that describe the amplitude of the waveform at each moment in time. The number of samples received per second is called the sampling rate. A common sampling rate for music is 44100 samples per second and is the standard for CD quality audio.

To process this data for spectral analysis, computers typically take a window of samples (for example, 1024 samples), do some processing, and then wait for the next window. Window size is important as larger windows contain more information about the signal but are received less often than smaller windows. With a window size of 1024, windows will be received every 23 milliseconds, whereas 8192-sample windows would be received every 186 milliseconds.

In order to gain information more often, sometimes a sliding window is used with a hop size parameter. For example, if the actual buffer size is 512 samples, the computer can save the last few windows and do calculations on a larger

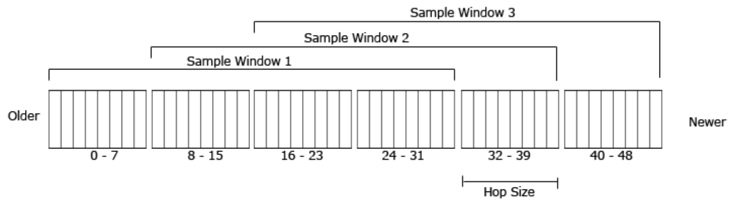


Figure 2.4: An example showing a hop size of 8 with an effective window size of 32

window of say 4096 samples. This introduces overlap which can help reduce artifacts and improve resolution in the frequency domain.

2.2.1 Detecting Pitch of Single Notes

There are two main approaches of detecting the pitch of a sound. The first is by analyzing information in the time domain (the explicit data contained in the samples), and other is by analyzing the frequency domain (the distribution of the different frequencies spread throughout the frequency spectrum). One popular time domain algorithm is called autocorrelation [28]. It looks for the highest peaks of each wave in the sample window, and measures the distances between them. With these distances, the period of the wave can be inferred, and the frequency can be detected. This method works well for individual pitches or notes, but when there is more than one note sounding at a time, the algorithm is unable to classify them.

To analyze a sample window in the frequency domain, a common technique is to run the samples through a function called the Fourier transform. This function takes in an array of amplitudes at discrete points in time, and outputs an array of frequency bins that contain the magnitude and phase of different parts of the frequency spectrum. By locating the bins with the largest magnitudes, it

MIDI #	Pitch	Freq (Hz)
40	E	82.4
41	F	87.3
42	F#/Gb	92.5
43	G	98.0
44	G#/Ab	103.8
45	A	110.0
46	A#/Bb	116.5
47	B	123.5
48	C	130.8
49	C#/Db	138.6
50	D	146.8
51	E#/Eb	155.6
52	E	164.8
53	F	174.6
54	F#/Gb	185.0
55	G	196.0
56	G#/Ab	207.7
57	A	220.0
58	A#/Bb	233.1
59	B	246.9
60	C	261.6
61	C#/Db	277.2
62	D	293.7
63	E#/Eb	311.1
64	E	329.6
65	F	349.2
66	F#/Gb	370.0
67	G	392.0
68	G#/Ab	415.3
69	A	440.0
70	A#/Bb	466.2
71	B	493.9
72	C	523.3
73	C#/Db	554.4
74	D	587.3
75	E#/Eb	622.3
76	E	659.3
77	F	698.5
78	F#/Gb	740.0
79	G	784.0
80	G#/Ab	830.6
81	A	880.0
82	A#/Bb	932.3
83	B	987.8
84	C	1,046.5
85	C#/Db	1,108.7
86	D	1,174.7
87	E#/Eb	1,244.5
88	E	1,318.5
89	F	1,396.9

Table 2.1: MIDI note names with associated pitch and frequency in Hz.

is possible to see which frequencies are present. In the case of a guitar note, the bin containing the fundamental frequency as well as any bins containing the frequencies of any harmonics will have significant magnitudes.

2.2.2 Detecting Pitch of Chords

These methods work well on individual pitches, but when multiple notes occur at the same time, they do not work for various reasons. First, measuring the highest peaks will only allow for the most prominent frequency to be found. Due to the nature of harmonics, looking at the Fourier transform output alone is not enough to figure out which notes are making up the signal. This thesis explores algorithms to overcome this challenge.

With standard six-string guitars, up to six simultaneous notes can be played at the same time. Many times, notes of the same pitch class are played on different strings but in different octaves. To form good estimations of which hand position and fingering are being used, we would like to know both the specific pitch class and which octave (determined by the pitch) each note corresponds to. There are a few different ways to go about detecting multiple pitches sounding at the same time.

First, special hardware can be employed to record the pitch of each string individually. With certain kinds of MIDI pickups, a guitar can output the signal of each individual string separately. The pitch can be computed from each string so that it is clear which note comes from which string, and the fingering of chords can be inferred unambiguously. The downside to this approach is that the guitar has to be specially built or modified to allow for the output as most guitars are not built this way.

Non-negative matrix factorization has been shown as an effective method to find the individual note components of a single audio signal by projecting the FFT onto pre-computed note templates. This allows it to work with a single microphone and no modifications to the guitar. This algorithm is described in detail in the following sections.

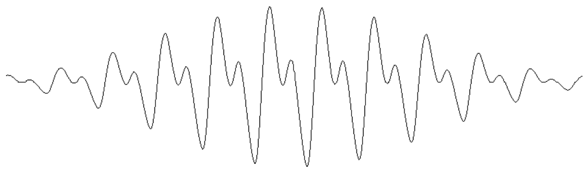


Figure 2.5: Physical wave form of a single guitar note

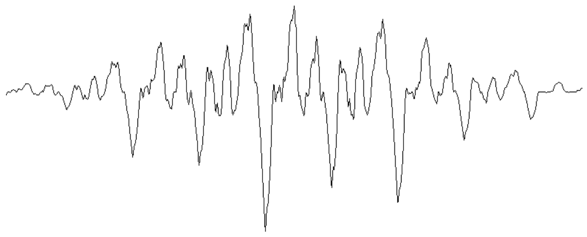


Figure 2.6: Physical wave form of guitar chord

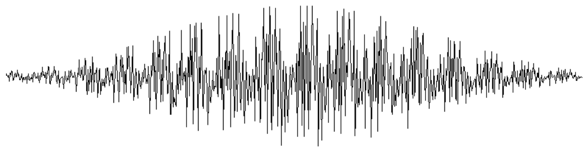
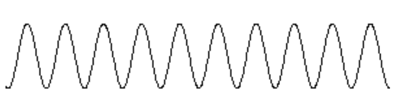
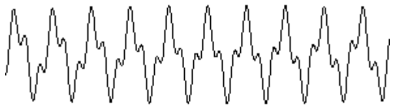
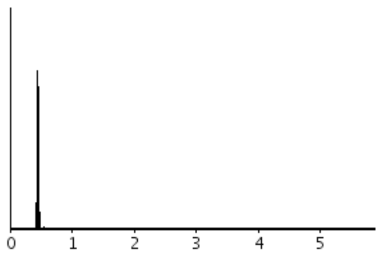


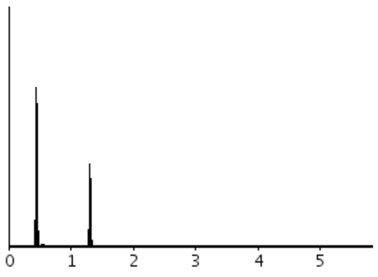
Figure 2.7: Physical wave form of a single note with distortion


(a) 440 Hz sine wave


(b) 440 and 1300 Hz sine waves



(c) FFT of one sine wave



(d) FFT of both sine waves

Figure 2.8: Wave forms of one (a) and two (b) sine waves, along with their resulting FFT outputs (c) and (d)

2.2.3 Shape of Guitar Notes

Time Domain

In the time domain, the note envelope is the shape of the amplitude or loudness of a musical note throughout its entire duration. The envelope is made up of four main parts. The first part, called the **attack**, is when the note is first picked, and can be seen as a sharp increase in amplitude from zero to the peak. After the attack, the **decay** is the period during which amplitude falls down from the peak to the main portion of the note called the sustain. During the **sustain**, the note gradually falls until the **release**, when the note is silenced or becomes inaudible. By paying attention to the relative amplitudes, note onsets can be inferred by sudden increases in amplitude. While this can suggest timing and temporal location of a note, it does not show us which pitch or how many notes are being played at a time.

Frequency Domain

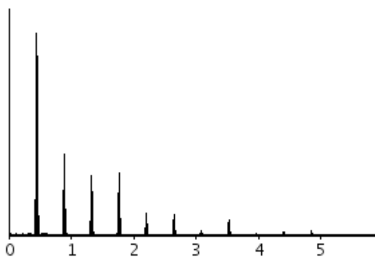
As mentioned previously, the Fourier transform, (specifically the discrete Fourier transform) takes in a vector of audio samples and produces a vector of frequency bands with both magnitude and phase information corresponding to different ranges of the spectrum. The equation for the DFT is shown in Figure 2.1.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi k \frac{n}{N}} \qquad k = 0, \dots, N-1 \tag{2.1}$$

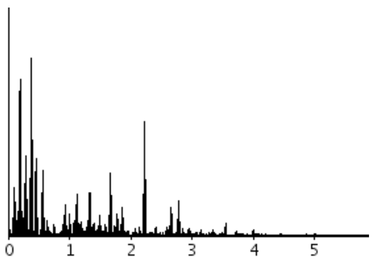
Because calculating the DFT is on the order of $\mathrm{O(N^2)}$, a fast Fourier transform (FFT) is used instead to calculate the same result in $\mathrm{O(Nlog}\,N)$ operations. When the FFT is taken repeatedly over short periods of time (often using overlapping windows), the result is called a short-time Fourier transform or STFT. The STFT can be visualized to show the change in local frequencies over time.

The Fourier transform result gives us information about equal spaced frequency bands from 0 Hz up to the Nyquist frequency which is equal to half of the sampling rate. The number of useful bands is equal to half of the number of samples in the input window. So, for example, if we are using a window of 1024 samples and using a sampling rate of 44.1 KHz, we will end up with 512 frequency bands spanning the total range of frequencies between 0 and 22050 Hz.

In addition to there being many notes with lots of harmonics, different notes may have overlapping harmonics. For example, the first four harmonics of the note A_2 are 110, 220, 330 and 440. For E_2, they are 82.4, 164.8, 247.2, and 329.6. At 44.1 KHz, it would take a very long window size to compute an FFT with high enough resolution for 329.6 and 330 to end up in separate bins. As more notes are sounding simultaneously, there is a larger chance of harmonic overlap. Also,



(a) Single note at 440 Hz



(b) Four notes played as a chord

Figure 2.9: Resulting FFT of real guitar notes

this shows that bandwidth of the FFT bins is important. For example, with a window size of 1024 samples, the bandwidth of FFT bins will be 43.1 Hz. The lowest notes on the guitar have fundamental frequencies that differ by less than 10 Hz, so at this resolution, many of the first harmonics will overlap with each other, making it difficult to distinguish between notes.

Another issue with the FFT output is that during the initial attack, there is a lot of noise associated when the finger or pick strikes the strings. Not until the actual sustain are frequency bins generally representative of the true harmonics of the sounding notes. While this attack period is very short, it must be taken into account.

Before FFT is computed, we apply the Hanning windowing function to the sample buffer. Window functions are used to scale the samples in the time domain, (often in the shape of a bell), and help to reduce edge effects that cause spectral leakage in the resulting FFT.

Chapter 3

The Algorithm

This thesis tackles the problem of detecting exactly which notes are being played on a guitar in real-time. While the Fourier transform gives us information about which frequencies are present in a signal, we need to do further analysis to determine which notes make up the signal, taking into account the characteristics of note harmonics. Non-negative matrix factorization has been shown to be a useful tool in sound separation, used offline in the context of drums [3] and guitar [17], as well as piano in real-time [8].

3.1 Non-Negative Matrix Factorization

Non-negative matrix factorization (NMF) is a family of algorithms to separate an input matrix V into two component matrices WH such that the terms in both are non-negative. In the context of detecting components of sound, the input matrix V is built from the STFT of the input audio. In other words, each column represents the FFT of a single window of samples. W represents a library of note templates where each column corresponds to the FFT representation of a certain note averaged over its sustain. H is then referred to as the activation matrix, with values corresponding to which note templates are contributing to the total sound at different points in time.

NMF is a good candidate for this problem because the magnitude of frequency bins is always non-negative. When more than one note is sounding at the same time, the resulting FFT contains magnitudes that are the sum of the FFTs of the individual notes (see Figure 2.8). This technique has been shown as an effective algorithm for polyphonic music transcription in a number of different studies [8].

3.1.1 How it works

To begin, an approximate factorization of the matrix is computed, and a cost function is used to evaluate it. Then the factor matrices W and H are modified with multiplicative updates until the cost function is minimized. The basic problem can be modeled by equation 3.1.

$$\mathbf{V} \approx \mathbf{WH} \tag{3.1}$$

In some studies, both W and H are initialized with white noise, and in others the template library is pre-computed. After each iteration, the cost is calculated, and the process continues until it converges or a certain number of iterations have occurred.

Cost functions

Following the work of Dessein et al., we use a beta-divergence cost function that interpolates between Euclidean distance and Kullback-Leibler divergence [8].

Template Note Library

We also used a fixed note template library W that is computed prior to running the actual algorithm. For each note, we record a short sample, and take the average of many FFT frames from the main part of the note sustain. In previous work, this template library needed to be built from samples recorded from the specific guitar for which it would be used for. In our work, created three template libraries, one with a nylon stringed acoustic guitar, one using an electric guitar with a clean tone, and one with an electric guitar with distortion.

When building the library, it is important that the guitar is in tune. When this library is used in practice, the guitar's tuning should match the tuning of the library.

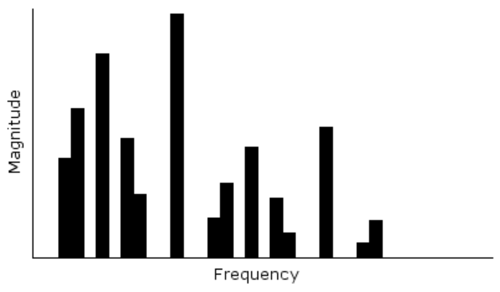
In section 4.4.1 we generate this library at run-time specifically for the current guitar, but unless otherwise mentioned we use a library computed from pre-recorded note samples as described here.

3.1.2 Real-time NMF

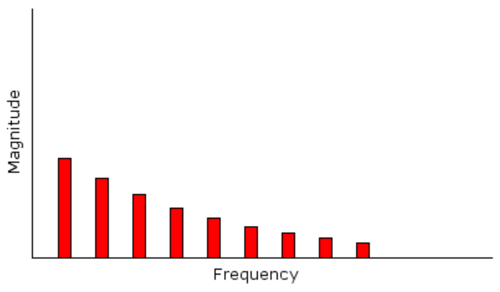
For this algorithm to work in real-time, the problem is simplified to calculate the activation row of each FFT frame as their corresponding sample window arrives each time. That is, as each column of v arrives, we calculate the row of H that will contain a note activation value for each template.

$$\mathbf{v} \approx \mathbf{W}\mathbf{h} \tag{3.2}$$

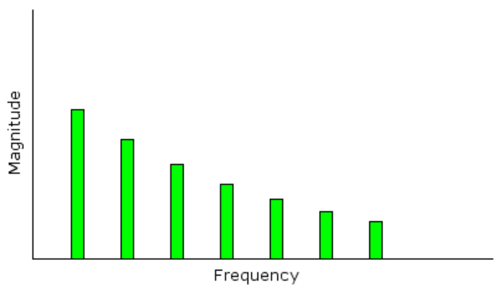
For example, say our input column v is represented by Figure 3.1 (a), and we have an input note template library of 30 templates, three of which can be seen



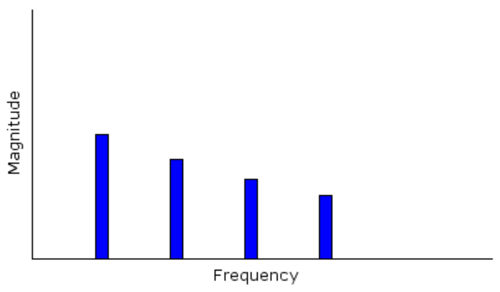
(a) Input FFT column V



(b) Note template 1



(c) Note template 2



(d) Note template 3

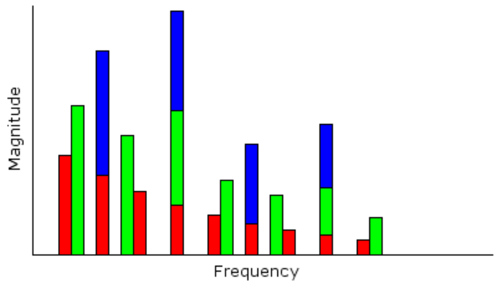
Figure 3.1: (a) Input FFT column V (b,c,d) FFT of note templates

in (b), (c), and (d). The goal is to figure out which combination of notes make up the input column when added together.

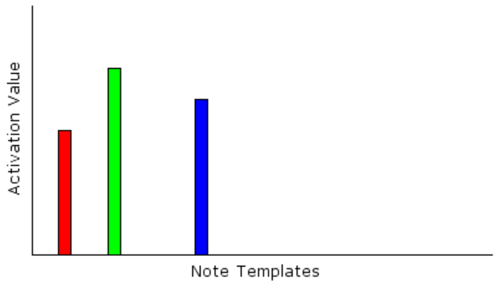
After running the updates on the column h until the cost function converges, we will get values corresponding to each note template.

3.1.3 Interpreting the NMF output

In practice, the NMF algorithm does not simply return exactly which notes are present. The results must be thresholded above some value to get rid of the noise of notes that are not actually sounding. Often, notes that are an octave



(a) The note templates that add up to the input FFT



(b) Note activations H corresponding to 3 specific note templates

Figure 3.2: (a) The combination of all the present note templates that make up the input FFT (b) The note activations (simplified, with no noise shown)

above an actually sounding note will show up as a false positive. Similarly, if the NMF cost never converges or ends up to be high, it indicates that the current results are not accurate, and are not recorded.

Because of errors like these, the NMF data is averaged over a specified number of frames. Another strategy to reduce error is to only recognize new notes when there is a spike in amplitude, indicating a note attack.

Chapter 4

Experimental Analysis

4.1 Overview

In this chapter we will discuss the approach we took of studying the different aspects of the NMF algorithm. In the first section we describe the main goals of the system. Next, we describe the system we built to evaluate the effects of different configurations of the program. We then describe the main tunable parameters and strategies we use to maximize the performance and accuracy of the results. Finally we present the results, and an example program utilizing our findings.

4.2 Goals

There are many different uses for a software tool that has the ability to accurately characterize guitar chords from a microphone input [7]. In this thesis, we present a tool to give a musician real-time feedback of which notes are being played in terms of a specific key and scale. To maximize the real world potential for such an application, we evaluate the program's accuracy, speed, accessibility, and usefulness, each of which are detailed in the following sections.

4.2.1 Accuracy

There are many different aspects of music that we aim to characterize from the input sound. First, we want to know the pitch class and octave of each individual note, including all notes that might be sounding at any given moment in time. While it is easy to hear when a note begins (also called the note onset), it is difficult to measure exactly how long a note lasts, and when it is no longer considered to be sounding. Due to this, we focus primarily on the location of the onset set of a note. In some cases, we estimate how long a note lasts, and which notes may be overlapping.

One of the common errors in polyphonic note classification is finding the correct pitch of a note in the wrong octave. Because notes in different octaves will have different locations on the fretboard, we take special care to find the correct octave of each note in a chord.

When only a single note is being played at a time, it is useful to know the exact frequency of the note and how far away it is from the true pitch frequency. Often times, especially in lead guitar, players will bend notes up to other notes, or use vibrato (bending the note up and down) as an effect. Unfortunately, the NMF algorithm we use does not account for these notes. The autocorrelation method described earlier has the ability to determine the exact frequency of a single note, but NMF would require many more templates in order to classify these notes.

4.2.2 Speed

In addition to being accurate, we want the application to work in real-time. In order to do this, we require an efficient method for analyzing audio samples. Much work has been done on recorded sound in [3, 12, 17], but there are fewer projects that can accurately do so in real-time [8]. We measure the time it takes to return a new data point in milliseconds, and aim to minimize this duration.

4.2.3 Accessibility

Once this tool is complete, we hope to maximize the ease in which real musicians can access it. We would like to be able to host it as a web application, and allow people to simply visit a website without downloading any additional software. The current implementation has been developed as a Processing application which can be exported as a Java applet.

4.3 System Design

4.3.1 Platform

While there are many different programming languages and platforms that support audio programming, we were focused on developing a program that would be able to be run inside the browser. Also, we wanted to something that could be eventually ported to mobile devices. While HTML5 has introduced new APIs for manipulating audio in the browser, there is not yet any common functionality to provide microphone access to the browser.

After trying out different environments, we decided to use Processing (www.processing.org), a Java-based programming language designed for quick prototyping. With Processing it is easy to create small sketches with both graphics and audio, and export them to applications or Java web applets. There are also numerous libraries available to simplify quick GUI creation, as well as audio programming, and music manipulation. In addition, because Processing is based on Java, it should be relatively straightforward to port the project to the Android operating system that is also based on Java.

4.3.2 Template Library Generation

To begin the analysis of NMF on detecting guitar notes, we first recorded sample notes for each of the different pitches on the first five frets of the guitar neck. Notes were recorded from a nylon string acoustic guitar, a clean electric guitar, and a distorted electric guitar. Following the work of [8], we built note template libraries in the form of matrices with rows representing different note templates. We used these note template libraries for the majority of this thesis except where we explictly discuss generating the template libraries at run-time.

4.3.3 Processing Audio Input

Within the Processing environment, we used a common audio library to capture raw microphone data. In most cases, we directed the input to the Line 6 USB device that connected the analog guitar input to the computer. This allowed us to have a clear signal without very much noise. We have also run tests while playing guitar into a regular laptop microphone with promising results.

The input samples are windowed and run through an FFT function also provided by the same library. Each output FFT is stored in a circular buffer in order to preserve past frames for visualization as well as future computation. In addition, note activations (the output vector h from the NMF algorithm) are also stored in buffers for the same purpose. These buffers are encapsulated as objects with member functions written to return both the raw values or averages over the n most recent values. This data is used in combination with amplitude information to detect note onsets and filter out noise.

4.3.4 Graphical Interface and Visualization

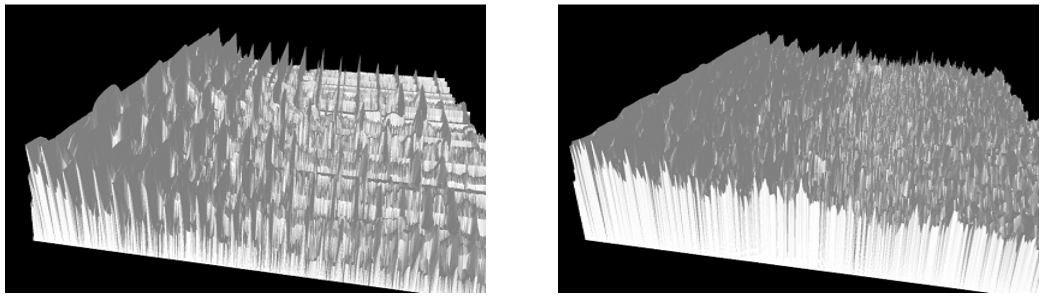
During this work, two main programs were developed. The first program was used for the majority of the experimentation, and was built to display different aspects of the audio in real-time using different visualizations. The other program is an example of an application of this algorithm to be used in a program to transcribe guitar audio in real-time.

FFT Visualization

Throughout this work, there were many times when it was useful to see visually what data was in the FFT output of both the note templates as well as the real-time sample windows. Both 2D and 3D visualizations were developed to get a better sense of how the different frequency bins reacted to different input over time, especially when different parameters were experimented with.

NMF Visualization

The most helpful view in evaluating the system subjectively was the 3D visualization of the NMF note template activations. By looking at a visual representation of the output, it was easy to manually tweak threshold values to more



(a) Single Note

(b) Chord

Figure 4.1: Real-time 3D visualization of FFT

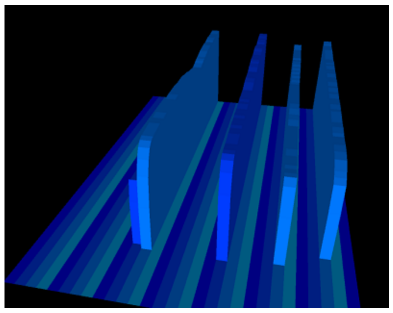


Figure 4.2: NMF result of playing a chord after thresholding

accurately see which notes were being played at any given moment. When viewed on the same screen as the overall window amplitude level, it became clear that a lot of noise and false activations were the result of initial note attacks.

Fretboard Display

When experimenting, it was important to make sure that the resulting activations were actually the result of the correct pitches being played. To do this, a fretboard display was created to indicate which frets and strings notes were played. Although we attempted to classify what string a given note was coming from, we were unable to do so accurately. This program assumes that the notes are played as close together as possible with the smallest distance between the

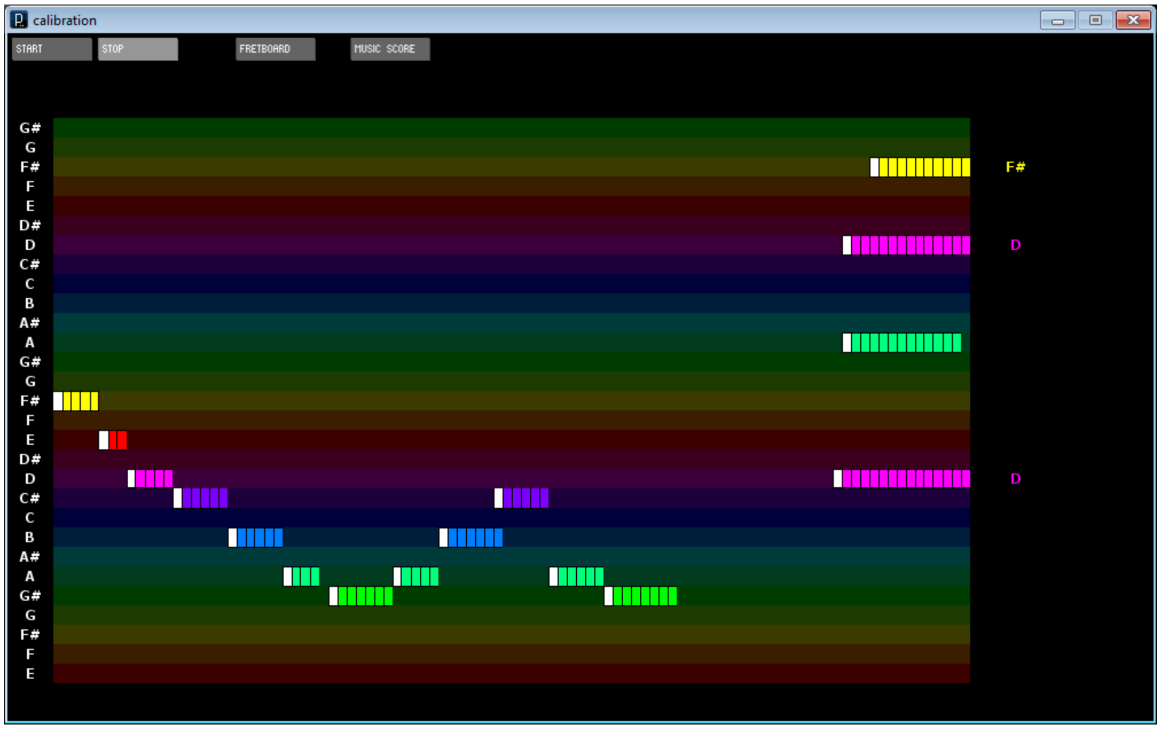


Figure 4.3: GUI Piano Roll Screen

lowest and highest fret numbers. If there are multiple places on the neck to play a chord, it is assumed to be played at the lowest possible position, or closest to the capo.

We do not infer anything about the practicality of the hand positions. Previous work has been done that stores a database of popular chord positions, and returns the best match for any set of notes [2]. Another group used a genetic algorithm to generate the easiest to play tab from a set of notes based on many different heuristics, such as distance between chords and finger positions [36]. This algorithm required knowledge of all the notes in a piece in advance, which is not possible in real-time computation.

Musical Score

The last view is the one that would be most useful to actual musicians. It shows the pitch of note activations, and displays the note throughout its main duration. If a specific music key is selected, it displays the notes that are being played and if they are or are not found in that key. Theoretically, the key could be inferred based on the notes that have already been recorded, but this is beyond the scope of our work.

4.4 Parameters

There are many different parameters that affect the performance of the NMF algorithm. To begin our work, we used a standard sampling rate of 44100 samples per second. For the majority of our tests, we had success with this rate, however experiment with other sample rates and discuss the motivations and implications of such later on. The next parameter is the window size of the incoming audio buffer. This determines the resolution of the FFT ouput. Because we desire both real-time calculations as well as larger windows, we slide the window by a hop size of somewhere between 256 and 1024 samples.

In terms of the NMF algorithm, there are also a number of different parameters. First is the number of note templates used. Too few note templates will result in ambiguous results and too many templates may cause the program to slow down and invalidate the results. Also, due to the range of guitar note frequencies, most of the interesting FFT data is located in the first 10-25 percent of the total number of bins. We have found that discarding portions of the FFT can significantly speed up the matrix multiplications while not degrading accuracy of

FFT Window Size	# of bins	Time Size (ms)	Bandwidth (Hz)
512	257	46.4	21.5
1024	513	92.9	10.8
2048	1025	185.8	5.4
4096	2049	371.5	2.7
8192	4097	743.0	1.3

(a) 11025 samples per second

FFT Window Size	# of bins	Time Size (ms)	Bandwidth (Hz)
512	257	23.2	43.1
1024	513	46.4	21.5
2048	1025	92.9	10.8
4096	2049	185.8	5.4
8192	4097	371.5	2.7

(b) 22050 samples per second

FFT Window Size	# of bins	Time Size (ms)	Bandwidth (Hz)
512	257	11.6	86.1
1024	513	23.2	43.1
2048	1025	46.4	21.5
4096	2049	92.9	10.8
8192	4097	185.8	5.4

(c) 44100 samples per second

FFT Window Size	# of bins	Time Size (ms)	Bandwidth (Hz)
512	257	5.8	172.3
1024	513	11.6	86.1
2048	1025	23.2	43.1
4096	2049	46.4	21.5
8192	4097	92.9	10.8

(d) 88200 samples per second

Table 4.1: Characteristics of FFT output with respect to sample rate and window size.

results.

The majority of the work done in the NMF algorithm is done during the convergence process which takes place over many iterations. There are a few different ways to determine if the matrices have converged, and it is important to limit the number of iterations to preserve the real-time aspect of the algorithm.

Once the iteration phase of the NMF has completed, the output activation matrix must be interpreted in an intelligent manner. To account for noise, we found that averaging the output of the past few data points improved accuracy quite a bit. We must determine the threshold for the minimum amount of an activation value to be considered an actual note, and not random noise or a different octave that is not actually being played.

The minimum amplitude threshold used to consider where note onsets occur must also be carefully chosen.

4.4.1 Run-time Calibration

In order to overcome the requirement to have pre-recorded note samples to build the note template library, we experimented with generating the library automatically for a specific guitar. To do this, we have the user play a note on one string, and use the autocorrelation algorithm to determine its pitch in the time domain. With that knowledge, we then measure the magnitude of the harmonics in the frequency spectrum. Once we have the general shape of the harmonics, we can recreate note templates for all the other notes.

In our experiments, the automatically generated note template library was fairly accurate, but often gave false positive for notes in the wrong octave. Although the algorithm did not perform as it did with the original template library, we believe that it would be possible to generate these libraries more accurately using more parameters than simply the shape of the harmonics. This would be a great feature, allowing any guitar to be used with the algorithm without having to record each and every note beforehand.

4.5 Performance Evaluation

To evaluate the performance of the algorithm, we recorded a number of audio samples of a single guitar playing single notes and chords, as well as songs in some cases. We then use these recordings as input to the real-time system. The output of the system is a list of notes occurring at different times within the recording, with associated pitches. We evaluate the performance of the system if it can process these files without audible delay, and by how many correct and incorrect notes the system outputs.

While building the program, it was easy see visually how well it was performing. After experimenting with different configurations and parameters, we did a final evaluation of how well the program performed on single notes, and chords of two, three, four, five, and six notes. To evaluate the accuracy so we can compare our results against similar projects, we use recall, precision, and an f-measure [8].

Recall is the total number of correctly detected notes divided by the total number of actual notes in the reference recording. It measures how completely **Precision** is the number of correctly detected notes out of the total number of detected notes. **F-measure** then defines the harmonic mean of the recall and precision.

Polyphony	Ref	C	M	EO	EN	Recall	Precision
1	29	29	0	3	2	1.00	0.94
2	48	47	1	3	0	0.98	0.94
3	48	47	1	5	0	0.98	0.90
4	64	59	5	4	1	0.92	0.89
5	65	61	4	4	3	0.94	0.90
6	66	65	1	4	1	0.98	0.93
Total	**320**	**308**	**12**	**23**	**7**	**0.96**	**0.91**

Table 4.2: Evaluation Results. Polyphony is the number of notes in each chord being tested. Ref is the number of total reference notes. C is the number of correctly detected notes. M is the number of reference notes that were missed. EO is the number of incorrect notes that were in the same pitch class as one of the correct notes. EN is the number of notes that were not in one of the correct pitch classes.

4.5.1 Test Environment

This evaluation was carried out using a pre-recorded note template library for a nylon string acoustic guitar. The window size used was 4096 samples with a 1024 sample hop length, with a sample rate of 44.1 KHz. Separate files were recorded containing one, two, three, four, five, and six notes at a time in each one.

This was run on a desktop computer with a six-core 3 GHz AMD Phenom II processor and 8 GB of ram, running Windows 7 Professional 64-bit. A total of 320 notes were recorded.

4.5.2 Analysis

From our test, we see that overall, the program performs well regardless of how many notes are played in the chord. These results are also biased by the fact that they were generated using the same guitar that generated the note templates, and most notes were played at a constant volume. Playing very quietly or loudly would result in less accurate note detection. We also note that recall is higher than precision in all cases, which indicates that we need to tweak the parameters to be more strict to avoid the false positive errors.

This test reflects the best performance of our implementation of the NMF algorithm. Tests with an electric guitar with a clean tone were almost as successful. Unfortunately, the algorithm performed very poorly on a distorted electric guitar input.

Chapter 5

Related Work

There have been many projects to analyze the musical qualities of digital audio. Most of them work offline, on already recorded files, and others are good at detecting monophonic information, but are unable to accurately classify polyphonic audio.

5.1 Existing Software

5.1.1 Capo

Capo is a commercial software package that aids musicians in learning how to play their favorite songs. It allows the user to slow the song down and maintain the original pitch of the song. In addition, it displays a spectrogram (frequency vs time graph), and allows the user to select and play individual parts of the song at a given time. Other features include vocal reduction, to minimize a particular voice in the song, and tablature inference, a feature to guess which frets would needed to play a particular piece of music.

While Capo is a powerful tool for what it is, it does not process audio in real-time. It takes in sound files (such as mp3) and does analysis on the file as a whole before outputting useful information. This thesis works to provide similar features as Capo, but in real-time.

5.1.2 Melodyne Studio

Melodyne is a similar program to Capo in that it analyzes musical data from an audio file. It also has the power to take apart chords and identify individual notes. Additionally, Melodyne allows for pitch correcting and editing after the fact. It has a high resolution in both the frequency and time domains. Unfortunately, like Capo, it does not run in real-time.

5.1.3 Tartini

Tartini is a program that does analyze music in real-time. It can correctly identify the pitch and loudness of notes as they are played. It also has a feature to analyze a few different types of note expression, such as vibrato and bending. The only thing that Tartini is missing is the ability to accurately identify individual notes from within chords.

5.1.4 Rocksmith

Rocksmith is a video game that allows the user to connect any real guitar to the gaming console. It then displays notes on the screen and the goal is to pick or strum the right notes at the right time. It is able to detect individual notes as well as basic chords to give the user feedback as to whether or not they played the correct note. Reviews of the game suggest that the chords are not always accurately registered and are usually made up of only a few notes (not using all six strings).

Chapter 6

Future Work

While this thesis sheds a light on what is possible with the NMF algorithm in terms of analyzing guitar notes, this is just a starting point for many different applications. As web browsers improve and new APIs become available, it may be possible to implement such analysis directly in JavaScript, eliminating the requirement for Java to be installed on client machines. Also, as the processor speeds and memory capabilities of mobile devices continue to improve, it is worth exploring the performance of this algorithm on a wide variety of platforms. Previous work has shown that this problem is highly parallel and can benefit from parallelization on platforms such as CUDA or OpenCL. Furthermore, with the ability to calibrate note template libraries at run-time, it is feasible that such an algorithm would work for many instruments other than guitar and piano. With a solid implementation of this functionality, many useful applications could be built to aid musicians in a wide variety of ways.

Chapter 7

Conclusion

In this thesis we explored how computers interact with sound and learned a lot about pitch detection of a single guitar. We were able to successfully employ a non-negative matrix factorization algorithm to correctly detect the pitches of chords with up to six simultaneous notes in real-time. With this analysis data, we created graphical visualizations both for experimentation purposes and for example tools for musicians. We were able to display not only which pitches were sounding, but also which frets on a guitar they might correspond to. This fret information is used to generate tablature as well as a graphical display of the fretboard as the guitar is being played.

The NMF algorithm is shown to be a good tool in the context of polyphonic guitar analysis, and we believe that further study will yield more accurate results at higher resolutions, opening up the possibilities for many useful applications.

Bibliography

[1] Ultimate guitar tabs, 2012.

[2] A. Barbancho, A. Klapuri, L. Tardon, and I. Barbancho. Automatic transcription of guitar chords and fingering from audio. *Audio, Speech, and Language Processing, IEEE Transactions on*, (99):1–1.

[3] E. Battenberg. *Improvements to percussive component extraction using non-negative matrix factorization and support vector machines*. PhD thesis, Masters Thesis, University of California, Berkeley, Berkeley, CA, 12 2008, 2008.

[4] E. Battenberg, A. Freed, and D. Wessel. Advances in the parallelization of music and audio applications. In *Proc. of the Intl Computer Music Conference, New York, USA*, pages 349–352, 2010.

[5] E. Battenberg and D. Wessel. Accelerating nonnegative matrix factorization for audio source separation on multi-core and many-core architectures. In *10th International Society for Music Information Retrieval Conference (ISMIR 2009)*, 2009.

[6] E. Benetos and S. Dixon. Polyphonic music transcription using note onset and offset detection. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 37–40. IEEE, 2011.

[7] M. Brooks. Developing visualization software for musicians. 2010.

[8] A. Dessein, A. Cont, and G. Lemaitre. Real-time polyphonic music transcription with non-negative matrix factorization and beta-divergence. In *Proc. 11th International Society for Music Information Retrieval Conference (ISMIR2010)*, page 2, 2010.

[9] J. Durrieu, B. David, and G. Richard. A musically motivated mid-level representation for pitch estimation and musical audio source separation. *IEEE Journal of Selected Topics in Signal Processing*, 5(6):1180, 2011.

[10] C. Duxbury, M. Davies, and M. Sandler. Separation of transient information in musical audio using multiresolution analysis techniques. In *Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-01), Limerick, Ireland*, 2001.

[11] D. Ellis. Extracting information from music audio. *Communications of the ACM*, 49(8):32–37, 2006.

[12] C. Févotte, N. Bertin, and J. Durrieu. Nonnegative matrix factorization with the itakura-saito divergence: With application to music analysis. *Neural Computation*, 21(3):793–830, 2009.

[13] D. Fragoulis, C. Papaodysseus, M. Exarhos, G. Roussopoulos, T. Panagopoulos, and D. Kamarotos. Automated classification of piano-guitar notes. *Audio, Speech, and Language Processing, IEEE Transactions on*, 14(3):1040–1050, 2006.

[14] T. Gagnon, S. Larouche, and R. Lefebvre. A neural network approach for preclassification in musical chords recognition. In *Signals, Systems and Com-*

puters, 2003. Conference Record of the Thirty-Seventh Asilomar Conference on, volume 2, pages 2106–2109. IEEE, 2003.

[15] R. Gang, G. Bocko, J. Lundberg, S. Roessner, D. Headlam, and M. Bocko. A real-time signal processing framework of musical expressive feature extraction using matlab. 2011.

[16] M. Goto. A real-time music-scene-description system: Predominant-f0 estimation for detecting melody and bass lines in real-world audio signals. *Speech Communication*, 43(4):311–329, 2004.

[17] S. Hazra. Automatic tablature generation for guitar.

[18] A. Klapuri. *Signal processing methods for the automatic transcription of music*. Tampere University of Technology Finland, 2004.

[19] A. Klapuri. Multipitch analysis of polyphonic music and speech signals using an auditory model. *Audio, Speech, and Language Processing, IEEE Transactions on*, 16(2):255–266, 2008.

[20] A. Klapuri, T. Virtanen, A. Eronen, and J. Seppänen. Automatic transcription of musical recordings. In *Consistent & Reliable Acoustic Cues Workshop, CRAC-01, Aalborg, Denmark*. Citeseer, 2001.

[21] E. Larson and R. Maddox. Real-time time-domain pitch tracking using wavelets. *Proceedings of the University of Illinois at Urbana Champaign Research Experience for Undergraduates Program*, 2005.

[22] P. McLeod. *Fast, accurate pitch detection tools for music analysis*. PhD thesis, PhD thesis, University of Otago. Department of Computer Science, 2009.

[23] P. McLeod and G. Wyvill. Visualization of musical pitch. In *Computer Graphics International, 2003. Proceedings*, pages 300–303. IEEE, 2003.

[24] P. McLeod and G. Wyvill. A smarter way to find pitch. In *Proceedings of International Computer Music Conference, ICMC*, 2005.

[25] L. Oudre, C. Févotte, and Y. Grenier. Probabilistic template-based chord recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, (99):1–1, 2010.

[26] A. Ozerov and C. Fevotte. Multichannel nonnegative matrix factorization in convolutive mixtures for audio source separation. *Audio, Speech, and Language Processing, IEEE Transactions on*, 18(3):550–563, 2010.

[27] A. Pertusa and J. Iñesta. Multiple fundamental frequency estimation using gaussian smoothness and short context. 2008.

[28] C. Roads. *The computer music tutorial.* The MIT press, 1996.

[29] M. Ryynanen and A. Klapuri. Polyphonic music transcription using note event modeling. In *Applications of Signal Processing to Audio and Acoustics, 2005. IEEE Workshop on*, pages 319–322. IEEE, 2005.

[30] M. Ryynanen and A. Klapuri. Automatic bass line transcription from streaming polyphonic audio. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 4, pages IV–1437. IEEE, 2007.

[31] M. Ryynänen and A. Klapuri. Automatic transcription of melody, bass line, and chords in polyphonic music. *Computer Music Journal*, 32(3):72–86, 2008.

[32] C. Schörkhuber and A. Klapuri. Constant-q transform toolbox for music processing. In *7th Sound and Music Computing Conference, Barcelona, Spain*, pages 3–64, 2010.

[33] K. Sommer and C. Weihs. Analysis of polyphonic musical time series. *Advances in Data Analysis, Data Handling and Business Intelligence*, pages 429–437, 2010.

[34] A. Stark and M. Plumbley. Real-time chord recognition for live performance. In *Proceedings of International Computer Music Conference*, volume 8, pages 585–593. Citeseer, 2009.

[35] M. Stein, J. Abeßer, C. Dittmar, and G. Schuller. Automatic detection of audio effects in guitar and bass recordings. In *Proc. of the AES 128th Convention*, 2010.

[36] D. Tuohy and W. Potter. A genetic algorithm for the automatic generation of playable guitar tablature. In *Proceedings of the International Computer Music Conference*, pages 499–502. sn, 2005.

[37] A. von dem Knesebeck and U. Zölzer. Comparison of pitch trackers for real-time guitar effects. In *Proc. 13th Int. Conf. on Digital Audio Effects (DAFx), Graz*, pages 266–269, 2010.

[38] G. Wang, R. Fiebrink, and P. Cook. Combining analysis and synthesis in the chuck programming language. In *Proceedings of the International Computer Music Conference*, pages 35–42, 2007.

[39] V. Zenz and A. Rauber. Automatic chord detection incorporating beat and key detection. In *Signal Processing and Communications, 2007. ICSPC 2007. IEEE International Conference on*, pages 1175–1178. IEEE, 2007.