# The QuickieComposer Android Application

A Senior Project submitted in partial fulfillment of requirements for the Bachelor of Arts Degree in Liberal Arts and Engineering Studies

By

Brigit Elizabeth Hawley

Liberal Arts and Engineering Studies Department

College of Engineering

California Polytechnic State University

San Luis Obispo

Fall Quarter, 2012

## Abstract

The QuickieComposer application is a program intended for use on Android Devices. Designed for the composer on the go, the QuickieComposer app is perfect for those “in the moment” compositions. In the following document, the functionality and development of the first version of the QuickieComposer application is explained.


## 1. Introduction

In present day music, composition is a very competitive art. With digital music and Internet streaming available at everyone's fingertips, the demand for new music is quite high. Each composer looks for a way to define themselves, above and beyond the current standard of music – they need to come up with more new melodies, ones that will appeal to modern audiences. Not all of us were born with every note already in our heads like Mozart and most of us are not as brilliant at devising themes as Beethoven. Looking for that perfect melody can be an immense challenge.

As a composer myself, I have a hard time remembering a melody that I think of while walking to class. I'll have the main theme and a counter melody picked out before I enter one of my college classes, but by the time I get home to write down the deliciously intricate melody, I'll have lost a few of the finer details. Wouldn't it be wonderful if I had something to remember the melody by? Something that is better than paper to notate my compositions. Something that didn't require me to sit down at a keyboard to plunk out the notes I have in my head. Something that I can carry with me on a day to day basis.

That was the inspiration behind the QuickieComposer Application. As an Android application, this program will be available in a composer's pocket at any time. All one needs to do is whip out your phone, open the app, sing a little melody and *presto*! You have your song written and saved for later access. In the following paper, I will present the design of this application, explain how the program functions and give an analysis of the application's accuracy.

## 2. The QuickieComposer App

The “QuickieComposer App” is an application designed to assist composers. In this prototype version of the QC, the application is designed for use on a home computer. The program uses whatever usable microphone it can connect to (built-in, USB, etc) and if the program does not detect a usable microphone, it will output an error message and exit. Figure 2.1 shows an image of the QuickieComposer.

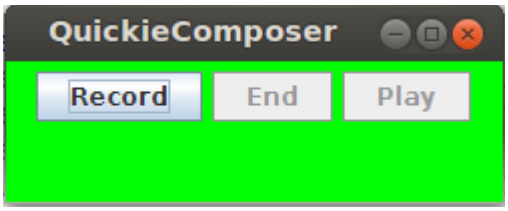

Figure 2.1

The interface is a fairly simple design with only three buttons. The 'Record' button starts the recording process of the application and disables itself once clicked. At any time, the user may press the 'End' button to stop the microphone from recording. Once there is a sound clip for the application to access, the 'Play' button will be enabled and the user may play back what has been recorded. The user may press the 'End' button at any point to stop the playback, although the application will *not* remember where it paused and will start the clip over if the play back button is pressed. At any time the user may record a new sound sample (by using the 'Record' button) and the new clip will overwrite the old. This simple design allows for easy usage by composers everywhere.

The simplicity of this design is due mostly to the audio API backing the QuickieComposer. The Java Sound API is a library designed for affecting and controlling the input and output of sound media. It is written in a way that is flexible for programming and promotes explicit control over the capabilities normally required for sound input and output. These characteristics make it perfect for writing a user friendly pitch detection program. There are dozens of helpful tutorials on the web about Java Sound API[1]. The QuickieComposer's interface is based on the Java Sound tutorials at Developer.com and the more in depth concepts behind the Java Sound library came from Java Sound Developers Guide. Both are cited in the Works Cited section.

For the QuickieComposer, the most important package in the Java API is the javax.sound.sampled (Java Plug). This package handles the digital audio data and is responsible for taking sound samples. 'Samples' are successive snapshots of wave data. In the case of audio, a microphone converts the acoustic signal into a corresponding analog electrical signal, and an analog-to-digital converter transforms that analog signal into a sampled digital form. Without these samples, the QuickieComposer would not have any sound data to work with.

The true functions of the QuickieComposer begin after acquiring sound samples. By using several of the object types defined in the Java Sound API, the QuickieComposer records and plays any audio passed into the microphone. To see a basic diagram of how this works, see figure 2.2.

1 Java Sound API Tutorial with Dr. Baldwin – See Further Reading Section

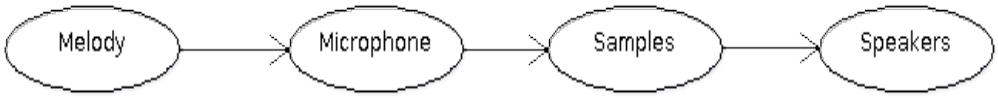


Figure 2.2

The recorded samples can be analyzed and transformed into the musical notation that musicians use (See Figure 2.3 for an example). Since Beethoven's day, musical notation requirements have changed; most publishers will not accept music that is not properly notated. The QuickieComposer app will be the future of music notation – accurate and precise.

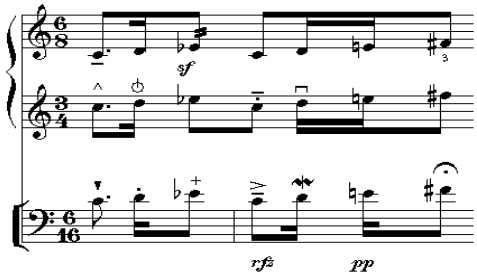

Figure 2.3

## 3. Design and Implementation

The concept behind the QuickieComposer is quite simple: write a program that records sound, analyzes it and then output the appropriate frequency data. On a more sophisticated level, the QuickieComposer program creates a graphical user interface (a GUI) and then allows a user to manipulate sound. He or she can record sound, play it back and analyze the audio recording. The user may stop the recording or playing processes at any time. Once playing, the user can see the analyzed data displayed in the terminal. Figure 3.1 shows a case diagram of how this process works.

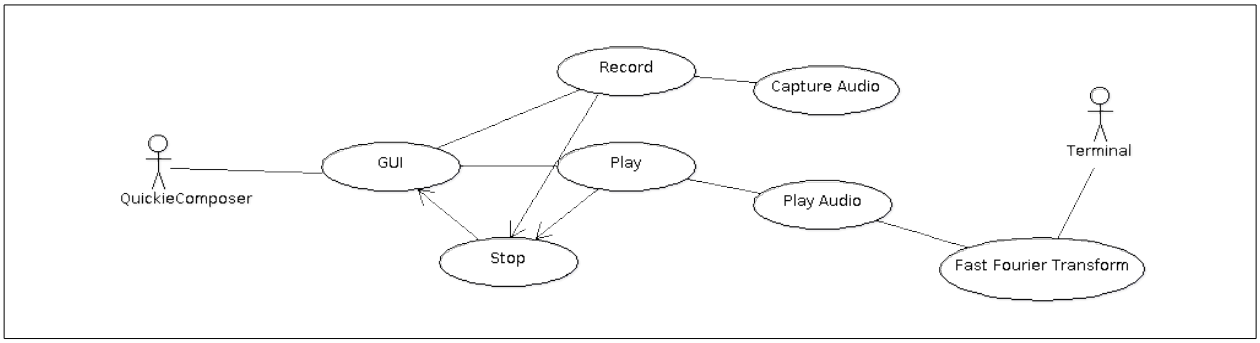


Figure 3.1

Figure 3.1 is a coding diagram depicting how the QuickieComposer functions. The application initializes the Graphical User Interface (GUI) for user input and sets up the program for recording. Once the recording button is pressed, the QuickieComposer opens an data input line, which is then used to acquire audio data from the microphone. This sequence of events occurs due to function calls within the QuickieComposer. The first function call is to the captureAudio() function. This function takes in no parameters and has a void return type. As a result of this function, audio input is stored into a source data line object (See Figure 3.2). The source data line is an object specific to the Java Sound API and records audio data in segments of bytes.

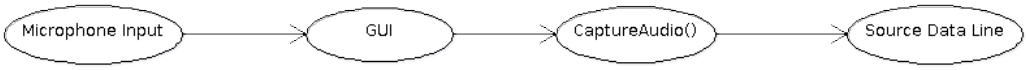


Figure 3.2

After the captureAudio() function has captured sound samples, the QuickieComposer waits for more user input. If the user decides to make another recording, the QuickieComposer makes another call to the captureAudio() function and deletes the previous set of samples. If the user decides to play the recorded data, the QuickieComposer then calls the playAudio() function. This function performs most of the work within the QuickieComposer. Data is read from the data line and converted into sound samples in a buffer array. Then the data array is passed into the constructor for a Fast Fourier Transform. The FFT converts the sound samples into pitch samples which are then printed out to the terminal (See Figure 3.3).


Figure 3.3

The current version of the QuickieComposer uses Java Swing to create the GUI. Java Swing is a Java library designed for making user interfaces. Most of the interface was designed out of a book called *Swing Hacks* (Marinacci). The implementation of this library allows for the QuickieComposer to interact with user input.

# 4. Experiments and Validations

Through thorough experimentation and development, the QuickieComposer App has evolved into a program with specific parameters. The application uses a 16 bit sampling rate because anything smaller cuts off many of the overtone frequencies, causing the sounds to lose richness. The microphone is set to mono channel because a SmartPhone's microphone is not

conducive to stereo sound; it is more resource-conservative if the app uses a single channel and creates a byte array of shorter length. The sample rate is currently set to 8192 samples – which means that the microphone is taking 8192 samples every second. This value was chosen because an FFT (Fast Fourier Transform) requires a sample rate that is two to a power and it is recommended by the Nyquist Frequency Theory that a sample rate should be two times the highest expected frequency. The highest note on a Grand Piano is about 4000 Hz and so:

$$4000 \times 2 = 8000 \qquad 8192 > 8000$$

The aforementioned reasons actually define the audio format used in this program. AudioFormat is a class defined within the Java Sound library. This class is used as the primary element in creating a new audio recording. The following is an example of a constructor for an AudioFormat object:

format = new AudioFormat(sampleRate, bitSize, channels, signed, endian);

The type of AudioFormat is important to the program because it specifies how sound data is stored and what ultimately can be done to manipulate that data. After designating the audio format file type, the QuickieComposer can begin recording and storing sound data.

All sound data is put into large byte arrays in an order that computers can understand. This order is based on the 'Endianness' specified in the AudioFormat constructor. The samples also have a designated signage – which means that the bytes are stored with either all positive values or positive and negative values. These various parameters imply the following: if you have a program running in Big Endian order and a signed sample rate of 16 bits, your computer will store sound data in a different way than if it was storing the bits as unsigned or if the order was Little Endian. The simplest way to decode the audio input data is to write a separate function that performs this task. Decoding the byte array will be slightly different for every application using the Java Sound API, but for the QuickieComposer Application, the following function is used:

```
int[][] getSamples(byte[] data) {
int low;
int high;
int sample;
int[][] toReturn = new int[channels][data.length / 2];
int sampleIndex = 0;

for (int aByte = 0; aByte < data.length; sampleIndex++)
  for (int channel = 0; channel < channels; channel++) {
    low = (int) data[aByte] & 0xff;
    aByte++;
    high = (int) data[aByte] << 8;
    aByte++;
    sample = high | low;
    toReturn[channel][sampleIndex] = sample;
  }
 return toReturn;
}
```

After the decoding process is complete, the decoded data is fed into an FFT. This must occur because computers do not store sound as frequencies, but as power samples taken over periods of time. To convert these samples into pitches, they must be converted to frequencies. Below are sample graphs of what the output of an FFT should look like. Taken from MatLab's built-in FFT, these images show 1.8 seconds of Queen's "Another One Bites the Dust." Figure 4.1 shows the song before being fed into an FFT (when the sound samples are still in the Time Domain) and Figure 4.2 shows the data after being converted into the Frequency Domain.

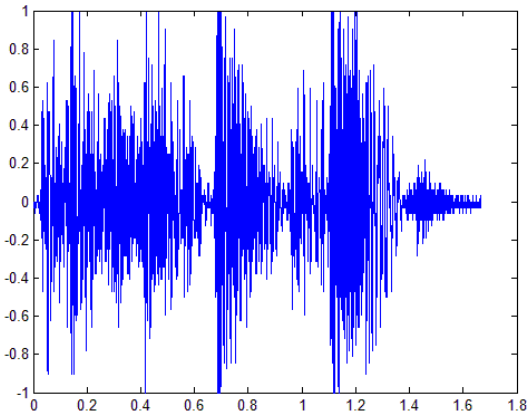


Figure 4.1

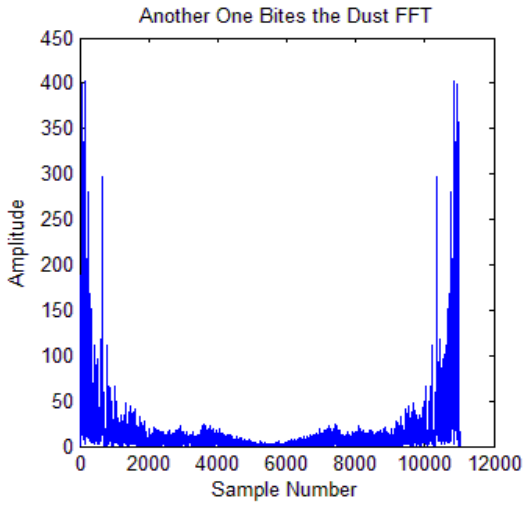


Figure 4.2

The Fast Fourier Transform is simply an algorithm used to convert wave data into another form of wave data. In this case, the samples were converted from the **Time Domain** into the **Frequency Domain**. This means that the samples are now readable as pitches, instead of pressure readings taken from a microphone. The source code for the FFT in the QuickieComposer Application is a program that was published on-line[2]. The program uses recursion to perform its analysis and requires complex numbers to function. Since the samples taken in by the microphone have no imaginary components, the QuickieComposer treats zero as being the imaginary part.

```
for (int ndex = 0; ndex < N; ndex++)
  data[ndex] = new Complex(samples[ndex], 0);
```

*Where N is the number of samples

The number of samples for an FFT is preferred as being to the power of two; although other numbers may be used if run with additional calculations. After the transform is run, the samples are returned from the FFT as complex numbers. They have an imaginary and real components. This poses a problem. How does one graph musical frequencies with an imaginary component? To finish the FFT functions, the QuickieComposer must calculate the magnitude of each sample to obtain the desired frequencies. The following function will create an array of frequencies that represent the audio data recorded by the QuickieComposer:

2 See Further Reading section: Sedgewick

```
public double[] getFrenquencies(Complex[] samples) {
    double[] toReturn = new double[samples.length];
    double real;
    double imag;

    for (int i = 0; i < samples.length; i++) {
      real = samples[i].re();
      imag = samples[i].im();
      toReturn[i] = Math.abs(Math.sqrt(real * real + imag * imag));
    }
    return toReturn;
}
```

It should be mentioned that there are other options for wave transforms. The Fast Fourier Transform is typically preferred however, due to its accuracy and fast performance time. FFTs take (LogN) * N arithmetical operations while many other transforms take O(N ^ 2). With their slower performance time and no more accurate results, other transforms just are not worth the comparative time cost. Most programs involving sound analysis implement FFTs for this reason.

## 5. Related Work

Although the marketplace does not have an application with the same function as the QuickieComposer, there are some apps that do similar things. There are several tuners available for musicians on the go. Most of these tuners do single pitch detection and determine if the note is slightly off or right on pitch. For instance, one of the most popular tuners on the Google Play Store has the following description:

Ultimate Guitar Tuner ($1.99) - “From the creators of Ultimate Guitar Tabs, one of the top Music applications in the Android Marketplace! Ultimate Guitar Tuner, which is part of the powerful Ultimate Guitar Tools suite, is now available as a separate application. Ultimate Guitar Tuner keeps your instrument in perfect pitch wherever you are. You can use your Mobile's built-in microphone, or tune by ear with our unique Brain Tuner. … Allows you to adjust your microphone sensitivity, and our ultra-high resolution display guarantees pitch perfect tuning. Includes the following accessories:”

- Mic Level Display
- Sleek interface
- New and improved pitch detection algorithm
- Alternate tunings
- Adjust microphone sensitivity
- Switch between linear and non-linear tuning scales
- Fine-tuning range increased
- Orchestral tuning

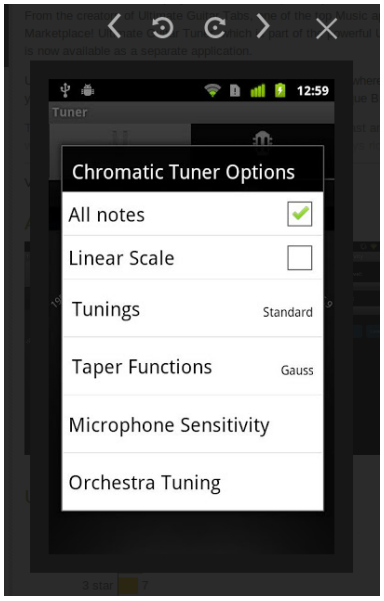


Figure 5.1


Figure 5.2

Figures 5.1 and 5.2 exhibit screen shots of the application's interface. This tuner, amongst others available on the Android market, represents the newest version of tuners available to musician. For years, people have used tuners to ensure their music is on pitch and audibly pleasing. This application uses a SmartPhone's built-in microphone and analyzes the sound in a similar fashion the QuickieComposer.

Another application, called the SpectralPro Analyzer ($5.49), presents analyzed sound in a visual fashion. This app gives accurate displays of high or even ultrasonic tones, environmental noise and rumble in a real-time setting. These spectrograms can be saved and exported as jpg or png files. The SpectralPro Analyzer also sports the following features:

- Selectable bandwidth up to 24 kHz
- Flexible amplitude
- Frequency mapping through touch and scroll
- Different update speeds
- Different color mappings.

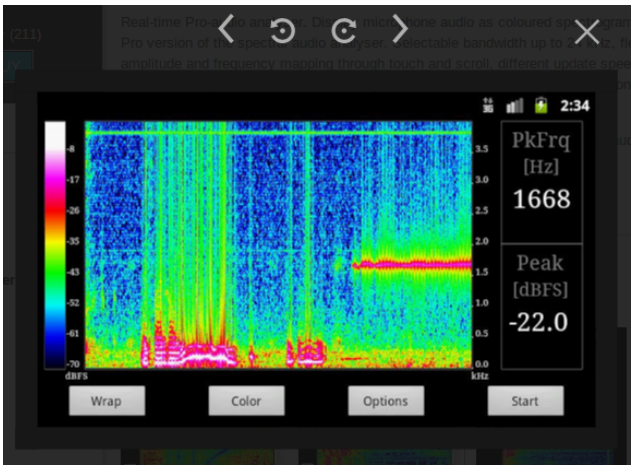


Figure 5.3

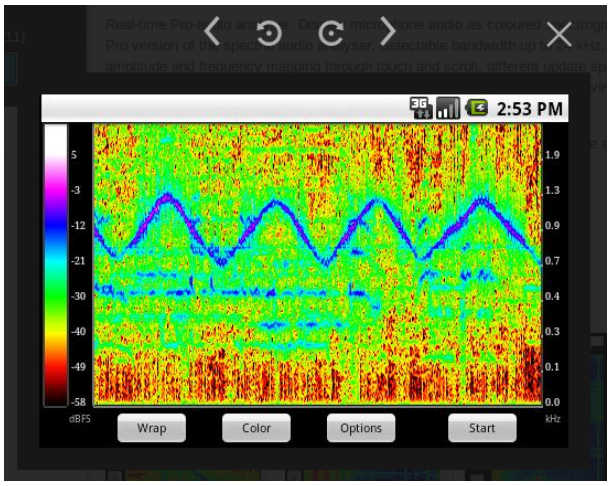


Figure 5.4

The SpectralPro Analyzer is another example of how Android applications analyze sound. Although this application produces graphs for analyzing data instead of listing the output, the basic analysis is similar to the QuickieComposer. Figures 5.3 and 5.4 show the SpectralPro analyzing sound. In comparison, this program is much better at depicting the sound waves – the QuickieComposer app is more specifically designed for analyzing whole musical phrases.

The next application to compare to the QuickieComposer is the AUDIOID Application. This application is probably the most different from the other apps mentioned in this paper. This program is meant to create music, rather than record it. Using loops and stock sounds, this application helps a user create a song by using pre-recorded sounds. With a TR-808 drum machine and the TB-303 bassline/groovebox, real-time filters, and randomized effects, this application is the perfect live DJ machine. Because AUDIOID  is a mobile application, this program is a great tool for the artist on the go, just like the QuickieComposer is meant to be. The AUDIOID application helps composers make music in another fashion – by providing an artist with lots of pre-made sounds.

As mentioned above, the QuickieComposer uses a Fast Fourier Transform to convert audio data into frequency data. Available on the Android Market are several FFTs. Below are screen shots of the most frequently downloaded FFT available.


Figure 5.5

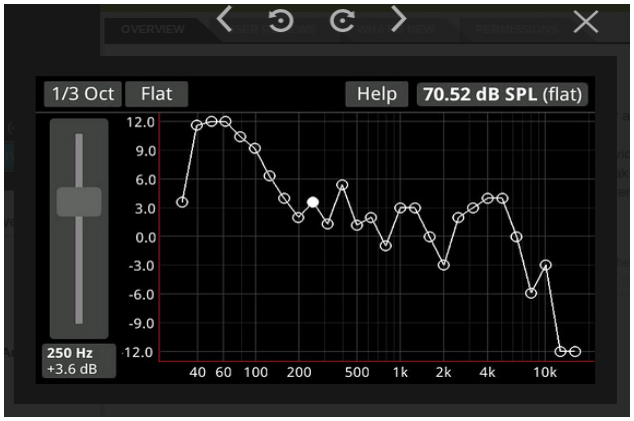


Figure 5.6


Figure 5.7

The last application to be evaluated is Shazam. This popular program is used to identify songs by people everywhere. The application records a melody through the microphone and then does a search to determine which song the melody is from. Other features of the application include:

- Save and listen again to songs
- See streaming lyrics in time to the music
- Buy tracks easily on Amazon MP3
- View extra content as you watch TV
- Watch music videos & concerts from YouTube
- Listen to your tagged music in Spotify
- Share on Facebook & Twitter
- Discover new music in Shazam Friends & Charts
- See when an artist is touring
- Use it when you do not have a signal
- When you see the Shazam prompt on TV tag for extra content
- Remember where you tagged a track with Location

Figure 5.8 and 5.9 show images of the Shazam interface.


Figure 5.8

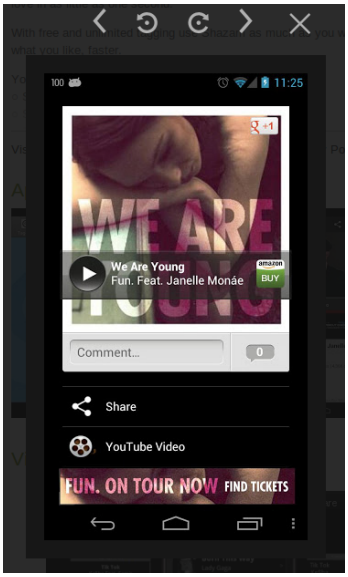


Figure 5.9

These features combined with Shazam's general purpose make it very important in this day and age. There are so many songs out there that listeners need help finding ones they like. The important connection between the QuickieComposer and Shazam is that it uses a recorded melody to help people with their musical ventures.

All of these applications are useful and have good ratings. They are also the most similar applications on Google Play to the QuickieComposer. As can be determined from the descriptions given, these applications do not have the same features as the QuickieComposer. It is true that several of the other apps analyze sound, but none of them focus on creating musical notation. The QuickieComposer will be the first application to venture into this avenue of sound analysis.

# 6. Conclusion

The QuickieComposer application is something that I hope will change how composers live their lives. Having a tool in one's pocket that could be used as easily as making a phone call could mean that musicians would have a better opportunity to be discovered. Something so simple could prove to be the most valuable tool for a busy musician. At this point in time, there is no application quite like it and so will earn a unique place in the SmartPhone marketplace.

The QuickieComposer makes recording easy – just press record and sing a little tune. The Java Sound API will take your recording and store it in the proper format. Press the play button – listen to your music play and watch the frequencies unfold. On a more in-depth level,

the QuickieComposer takes samples of audio data, stores them in the appropriate data type and then passes those samples to be analyzed. From there, the Fast Fourier Transform takes over and uses the recorded samples to create the musical notation needed for a beautiful composition.

In this paper I have gone over all of these features and the logic necessary for the QuickieComposer to work. The major concepts to take away from this project include:

- How computers record and store sound samples
- The Java Sound API is one of many coding libraries designed for sound programming
- A basic understanding of the Fast Fourier Transform and its purpose
- The functionality of the QuickieComposer Application

## 7. Future Work

With the conclusion of this project, the QuickieComposer Application is still not quite finished. The Fast Fourier Transform does not function the way it should, and I need to figure out how that can be fixed. Once the FFT is complete, the development of the rest of the QuickieComposer can begin. This, of course, means more research into the field of wave conversion. At this moment in time, I am working with several individuals in an attempt to further my understanding of the FFT.

The next step after fixing the FFT will be to write the notation software. This will entail accounting for various inputs from the user. The QuickieComposer might become more dependent on user input. This could include the user specifying a tempo for the song, or a tonic key. With this given information, taking the frequency data and mapping it to musical notation will be fairly simple. There are other ways to determine these specifications without inconveniencing the user, but they may not be as accurate. For instance, the QuickieComposer could run algorithms to find the most used pitch, or perhaps map the used pitches to a database listing of musical keys. Either way, the program will need an updated user interface and perhaps additional features. Testing will follow either implementation.

The final step of in the QuickieComposer project will be converting the program to a mobile platform. As mentioned above, the current QuickieComposer uses Java Swing to create the Graphical User Interface. Most mobile devices do not support Java Swing. Eventually, the QuickieComposer will become a program that uses Android SDK – the Java library for Android Development. Once the QuickieComposer has been updated to work with Android code, it shall be uploaded to the Google Play Store.

## 8. Further Reading

If you found the subjects in this paper to be particularly interesting and wish to find more please consider the following resources:

- Java Sound API Tutorial with Dr. Baldwin: These tutorials are the primary source for the QuickieComposer App's interface:
  http://www.developer.com/java/other/article.php/1565671/Java-Sound-An-Introduction.htm

- The Discrete Fourier Transform: This book is free and is excellent for understanding the FFT and Fourier Analysis on a simple level:
  http://www.dspguide.com/ch12/1.htm

- FFT.java: This is the code I used as the base for my FFT:
  http://introcs.cs.princeton.edu/java/97data/FFT.java.html

- For information on how to create great graphical user interfaces using Java Swing:
  *Swing hacks* by Joshua Marinacci and Chris Adamson

- For an introduction to general sound programming:
  *The Audio Programming Book* by Richard Charles Boulanger

## 9. Special Thanks

I'd like to mention the help of the various individuals who have helped me on my quest to understanding the Fast Fourier Transform:

Dr. Frederick Kitson
Craig Leeds
Larry McCrigler
Jack Cashin
Dr. Matthew Moelter
Dr. Paul Choboter

Thanks for taking the time to speak with me!

# Works Cited


"Android Marketplace." *Google Play*. N.p., n.d. Web. 12 Nov. 2012. <play.google.com/>.

Baldwin, D.. "Java Sound, An Introduction - Developer.com - Developer.com." *Developer.com: Technical Information for Software Developers*. N.p., n.d. Web. 23 Sept. 2012. <http://www.developer.com/java/other/article.php/1565671/Java-Sound-An-Introduction.htm>.

Boulanger, Richard Charles. *The Audio Programming Book*. Cambridge, Mass.: MIT Press, 2011. Print.

"Capturing Audio (The Java Tutorials Sound)." *Oracle Documentation*. N.p., n.d. Web. 3 Oct. 2012. <http://docs.oracle.com/javase/tutorial/sound/capturing.html>.

"Fast Fourier transform - Wikipedia, the free encyclopedia." *Wikipedia, the free encyclopedia*. N.p., n.d. Web. 13 Nov. 2012. <http://en.wikipedia.org/wiki/Fast_Fourier_transform>.

"Java Sound Developer Guide." *Oracle Documentation*. N.p., n.d. Web. 28 Nov. 2012. <http://docs.oracle.com/javase/1.4.2/docs/guide/sound/programmer_guide/contents.html>.

"Java Sound Resources: Examples." *Java Sound Resources*. N.p., n.d. Web. 15 Oct. 2012. <http://www.jsresources.org/examples/index.html>.

Marinacci, Joshua, and Chris Adamson. *Swing hacks*. Beijing: O'Reilly, 2005. Print.

Sedgewick, Robert, and Kevin Wayne. "FFT.java." *Introduction to Programming in Java: An Interdisciplinary Approach*. N.p., n.d. Web. 28 Nov. 2012. <http://introcs.cs.princeton.edu/java/97data/FFT.java.html>.

Smith, Steven W.. "Real DFT Using the Complex DFT." *The Scientist and Engineer's Guide to Digital Signal Processing*. N.p., n.d. Web. 3 Nov. 2012. <http://www.dspguide.com/ch12/1.htm>.

"Stack Overflow." *Stack Overflow*. N.p., n.d. Web. 17 Aug. 2012. <http://stackoverflow.com>.