**Audializer: Converting Pictures to Sound**

Signals and Systems: Project Final Report
Brendan Ritter, Brendan Quinlivan, & Greg Edelston
Spring 2013

## Project Overview

The goal of our project was to create an image-to-audio converter. We set out to develop a program that took in an image file as input, and produced an audio file which, when analyzed through a spectrum analyser, showed the original image. Ideally, the program would have produced wonderful-sounding audio.

In order to accomplish this task, our team initially set out to develop the entire program in MATLAB. However, after running into some issues with MATLAB's sound packages, we decided to complete the image analysis in MATLAB, and then call a Python script from MATLAB to complete the sound generation.

With this project, our team goals centered around developing an exciting application of signals and systems which would deepen our intuitive understanding of the content. This was especially important to us because the focus of the earlier portion of the class was on theory. Additionally, in this project we wanted to draw connections between signal processing and other disciplines (in this case we chose music).

## Image Analysis

Our program begins by taking an image into MATLAB in any of several different file formats, and converting that image into greyscale. After this conversion, the image is divided into vertical slivers, each one pixel wide. An example of this can be seen in Figures 1 and 2.

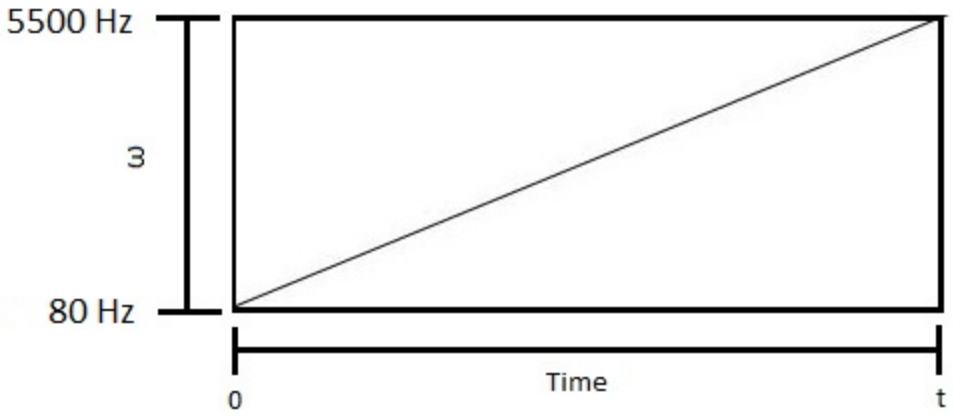


Figure 1. A greyscale image of a diagonal line where the vertical position of each pixel and the greyscale value correspond to the frequency and amplitude of the sound waves produced. Additionally, the horizontal position of each pixel determine the time at which the sound wave occurs in the sound clip.

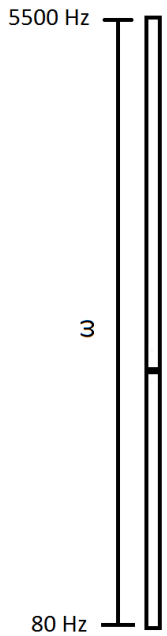


Figure 2. A vertical sliver of the diagonal line found in Figure 1 that is one pixel wide (image not to scale).

With the photo divided into slivers, the program maps each pixel in that sliver to a log scale of frequencies our laptop speakers could produce (80 Hz to 5,500 Hz) based on the pixel's height in the sliver. It is mapped such that the pixels at the bottom of the image correspond to low frequencies and those at the top correspond to high frequencies.

Finally, each pixel not only has an associated frequency based on its vertical position in the image, but also an associated amplitude based on the greyscale value of the pixel. Darker pixels denote higher amplitudes, while lighter pixels are lower amplitudes. For each sliver the program maps the frequencies to the corresponding amplitudes as shown in the graph in Figure 3.

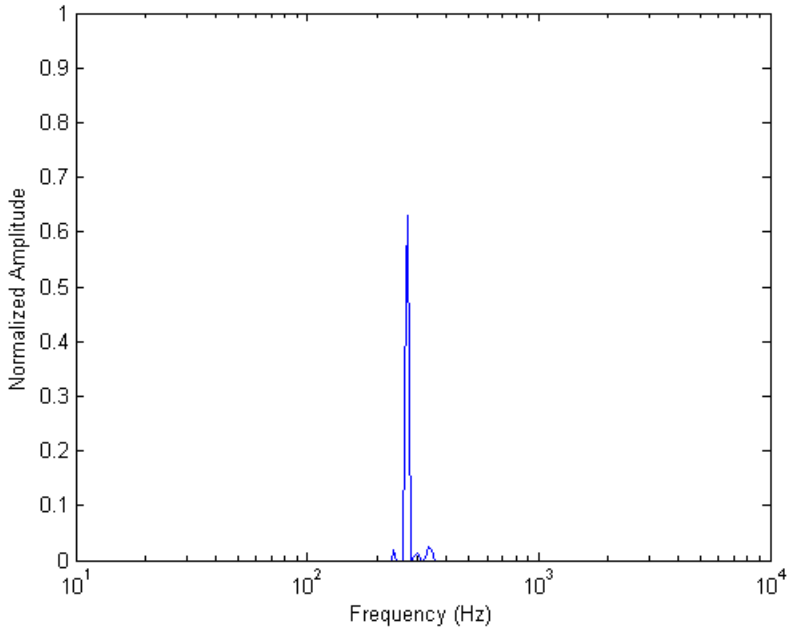


Figure 3. The normalized amplitudes and frequencies for the sliver shown in Figure 2. Notice that at approximately 250 Hz there is a spike in the amplitude which corresponds to the black pixels found in Figure 2.

Thus, the program has broken each vertical sliver down into a series of frequencies and corresponding amplitudes.

**Sound Generation**

After each sliver is analyzed, a list of all of the frequencies and amplitudes for all of the slivers is passed to a Python script through a CSV file. Each sliver is then made into a python dictionary of frequencies and their associated amplitudes.

A sine wave in time (with arbitrary duration) is generated for every frequency and amplitude in this dictionary with the incorporation of a frequency scaling constant. Thus, the outcome of this process is multiple sine waves. These waves are then added to make a chord and normalized based on how many sine waves were added so the amplitude doesn't get distorted. The result is a sound byte representing one particular sliver of the image.

When this process is repeated for all the slivers in the picture and their corresponding sound bytes are concatenated, an audialized version of the picture is created.

**How we used Signals and Systems**

When we analyze an image, we create a signal in the frequency domain which must then process into the time domain. However, since we are also operating in a discrete space, the

signal behaves as a sampled function with deltas as the values at each point.

We use the linearity of the Fourier transform to realize that the inverse transform of the sum of impulses is equivalent to the sum of the inverse transforms of the impulses.

We also realize that in order for the time-domain signal to be real, the frequency-domain signal must be symmetrical across the y-axis; therefore, we mirror the frequencies. For a given frequency $\omega_0$, this creates the frequency signal $\delta(\omega + \omega_0) + \delta(\omega - \omega_0)$. From Signals and Systems, we know that this equation is the Fourier transform of a cosine function; therefore, each impulse in the frequency domain maps to a sinusoidal function in the time domain.

It should be noted that a sine function can be interchanged with a cosine function, since they equivalent, save for a phase shift.

Signals and Systems helped us to understand what was going on behind the scenes. It would have been possible to blindly accept that a sine wave produces a pure tone and therefore makes sense to produce. However, using our accumulated knowledge from the semester, we understood everything that happened, and were able to produce music with full knowledge of why it works.

**Conclusions**

Overall, we were capable of making a picture to sound converter. However, there were several areas of our project that could have been improved.

One feature that would have made our project more cohesive would have been only performing operations in MATLAB. We decided originally not to do this due to complications with the sound producing abilities of MATLAB. However, given more time we could have straightened these problems out.

In addition, another option available to us was to use a piece of code created in MATLAB that produced guitar-like notes using some sort of dynamic filter. Although effort was put into deconstructing its method of sound creation, our current method was finalized before we could fully understand the guitar code. Thus, a further deconstruction would have allowed us to stay in the MATLAB domain, and include even more signals and systems.

A further area of investigation could have been the capabilities of our laptop's speakers. It was at times unclear whether our programming methods were incorrect, or whether we were getting distortion by running up against the hardware limit of our speakers. A further look into the hard limits might have actually validated previous MATLAB attempts to make sound.

**Areas for Future Work**

There is the potential for a lot of future work on this project. One potential improvement could be converting the frequencies to a pentatonic scale. This would have the benefit of making every image sound pleasing. However, this modification would make the process irreversible because there would no longer be a one-to-one mapping of vertical position to frequencies.

Along those same lines, a future project could be reverse the audialize process and make a picture out of a sound file. Based on our current program it would be trivial to convert the amplitudes and frequencies associated with each pixel back into an image. However, the process of dissecting the constructed audio file into these amplitudes and frequencies associated with each pixel would require a significant amount of work.

We can also foresee applications for our software with encryption. If there is a particular algorithm for mapping a given pixel to a frequency, amplitude, and time -- more complex than a simple logarithmic mapping -- then an image could be securely encoded into a sound file.