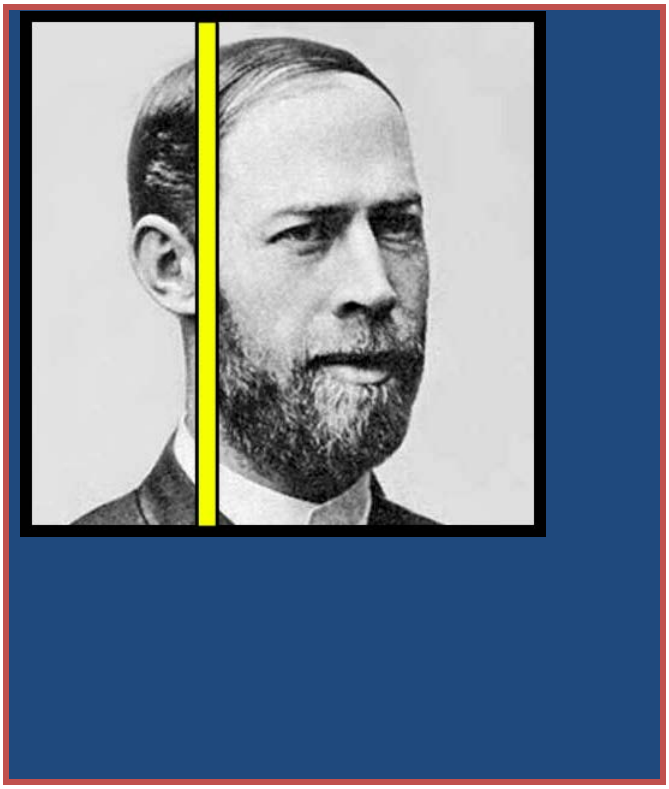


5/9/2014

Imajadio

COSC 4P98 Project

Authored by: Jakub Subczynski

COSC 4P98 Project – Imajadio

Contents

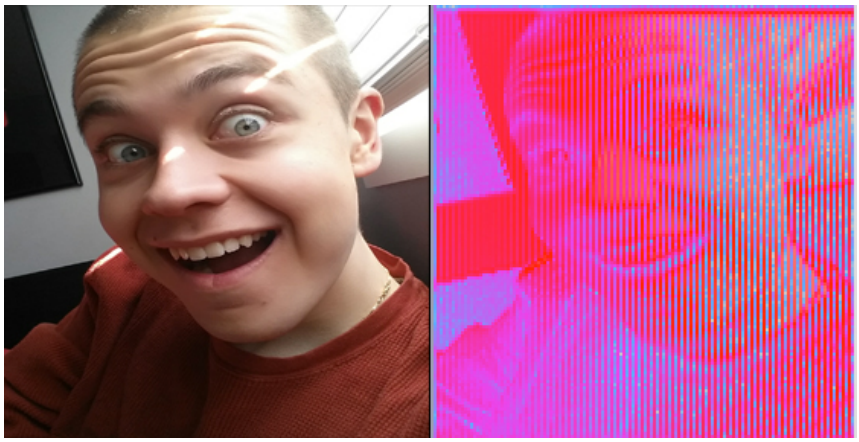
Description

Imajadio is a proof-of-concept android application that bridges the gap between two senses; sight and sound. The name comes from a unity of the words “Image” and “Audio” because that is exactly what our application does, unifies the two senses. The user can take a picture using their camera, or upload a previously captured image and our app will convert it into audible frequencies that relate to the properties of the picture. Pictures that are very different from each other will yield very different sounds and pictures that are similar will yield a very similar sound. This conceptual model is easy to understand and the user will be able to conceptualize this easily. However, the actual audio that is produced is difficult to conceptualize because our brains are not programmed to see sound. We will further get into the specifics of how we decided to map the properties of a picture into sound as well as parameterization and features in later sections of this document.

Background

Algorithm Planning

In order to be able to bridge the gap between the two forms of media, we treated each pixel in a column as a unique harmonic and sum them up via a reverse DFT. By doing this, it gives us a fairly lossless conversion into audio. What this then means is then is that we should be able to convert back to an image in order to reproduce this fairly well. Due to the following methods/formulas that were used in order to calculate the frequency and amplitude of each harmonic the conversion back would only produce a gray scale image of the original. To demonstrate the tight coupling between image and audio in this pixel-to-harmonic conversion technique, the image on the left is the original and the image on the right is a spectrogram generated using Audacity of the audio produced:



In order to generate the frequency of a particular pixel, we formulated the equation:

$$\boldsymbol{f(x) = (h - x) * F/(h + 1)}$$

Where: f = frequency of a pixel (Hz)

x = pixel's vertical offset (from top --> down)

h = height of image (in pixels)

F = maximum frequency

This formula spread the frequencies that we are using evenly throughout the image. Also, due to its generality, it will work for any number of frequencies spread across any number of pixels.

In order to calculate the amplitude for a particular pixel, we formulated the equation:

$$\boldsymbol{a(x) = \frac{[r(x) + g(x) + b(x)]}{765} * M/h}$$

Where: a = amplitude of a pixel

x = pixel's vertical offset (from top --> down)

h = height of image (in pixels)

M = maximum amplitude or resolution of audio

Here, it can be seen that we calculate the amplitude's strength using a factor how much red, green and blue is in the pixel. Using that factor, we multiply it by *M/h* (that determines the maximum amplitude that a single pixel can have).

Algorithm Implementation

Due to the thorough planning stage, actually implementing the algorithm was quite smooth. The Imajadio class receives the image to convert and the duration of each column. For each column, the required number of samples is produced using the summation of each harmonic (i.e. reverse DFT).

That said, we did come across one unexpected issue. We did not take into consideration the possibility for beating. Beating is the behavior in which two pure tones constructively and destructively interfere with one another. This creates a very "choppy" sound through the audio when two similar frequencies are played at the same time.

In order to try to combat this, we chose to skip ever 20th pixel in the image. Doing this spaced the pixels out more, which reduced the amount of beating observed.

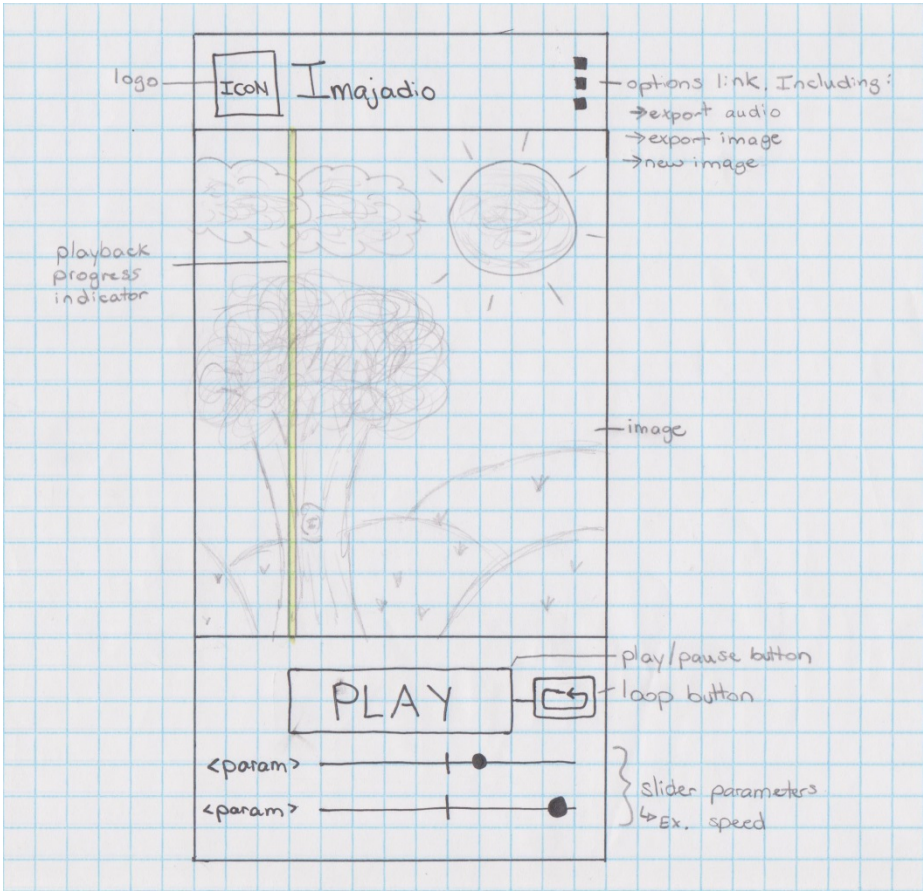
Implementation

Platform

The android platform was used because it is perfect for our application since smartphones have all of the hardware needed to give the user a positive experience. The fact that they have cameras built in allows the user to take a picture of something that they are physically experiencing and then HEAR what it sounds like. This is an abstract type of thinking that not many people can experience instantly. Smartphones also have a speaker built in so the feedback from the application can be heard instantly and on the go.

User Interface

The following is an early mock-up of what we wanted the user-interface to look like:



We decided to make most of the screen be taken up by the picture itself. This allows the user to see exactly what is being converted into sound. As seen in the picture we decided to implement a playback progress indicator. As our algorithm dictates, the picture is broken down by one sound per column and then played from left to right. The playback indicator shows the user what column is being played at that exact moment. This assists in the user's comprehension of

the picture being mapped to sound (ie. When the vertical pixels in the picture change dramatically, so does the sound.)

Underneath the image are all of the controls for the app. Of course the main control is the play button. This button actually has three uses: Convert, Play, and Stop. Our conversion algorithm is processor heavy so the time it takes to convert the picture is quite high. After the picture is loaded in and the parameters are set by the user the user would press “Convert” which starts running the algorithm. Once it is done, the convert button changes into a “Play” button so the user can listen to the audio. During playback, the “Play” button changes into a “Stop” button to stop the playback. This gives the user the ability to change the picture and parameters as he or she pleases to yield a different sound.

The main parameters are of course the picture itself, as well as the duration that each column of pixels is played for. This was implemented as a slider underneath the play button to make it easy for the user to change the value.

Features

We wanted our application to be as robust as possible so that the user would be able to use our application just as a starting point in their exploration of image-audio conversion. We added the ability to export their output so they can use the plethora of other audio tools available online to continue research.

Export Picture: This feature allows the user to export the picture they used into the phone’s internal memory so that it can be saved and viewed later. It is saved to a new folder called “Imajadio” which is stored on the root of the internal SD card of the phone.

Export Audio: This allows the user to export the actual audio that was yielded after running our algorithm. This is also stored to the Imajadio folder with the filename IMAJADIO_*timestamp.* The format is an uncompressed wave file so it is easy to load into an audio application like audacity to explore the output of the algorithm and gain a better understanding.

User Manual

On Launch

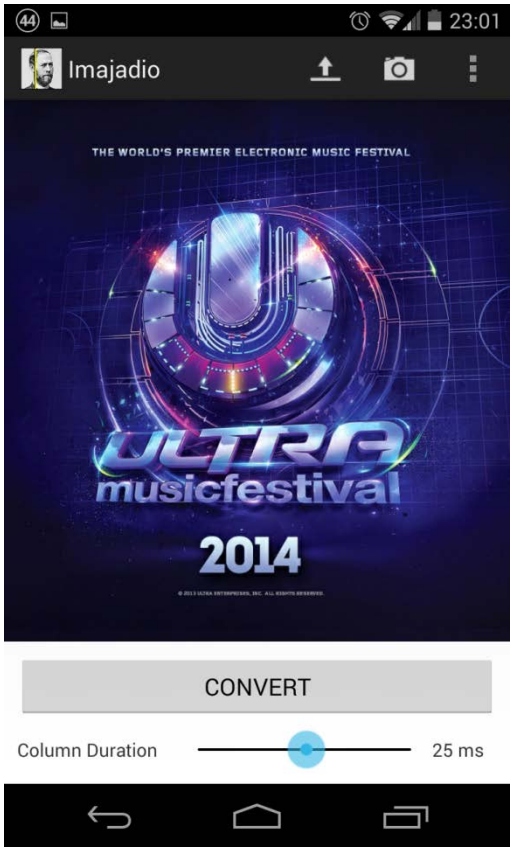
On launch, the user is given the main screen with a default image pre-loaded. One can use this image or they can upload their own as explained below. The parameters are set to the default of 25 ms.

Taking a Picture

A picture can be taken using the built in camera of the phone simply by pressing the Camera icon on the top of the application. Depending on your version of android this might also be found in the settings menu. After clicking on this you will be taken to your stock camera application. Just snap the picture, press accept, and your picture will be loaded into the Imajadio application. Be aware that the application crops the outer edges of the photo so it can fit perfectly in the viewing window.

Loading a Picture

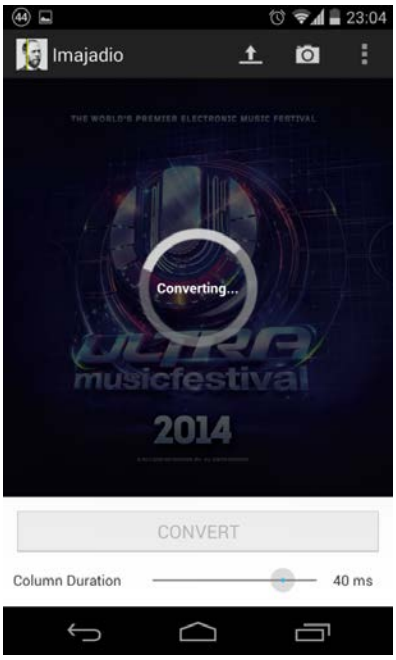
The user can choose to upload a picture from their gallery/camera roll. The user must choose the Load icon in this case which is found in the same place as the camera icon. They will be directed to their gallery and can load a picture in from there.



Converting to Audio

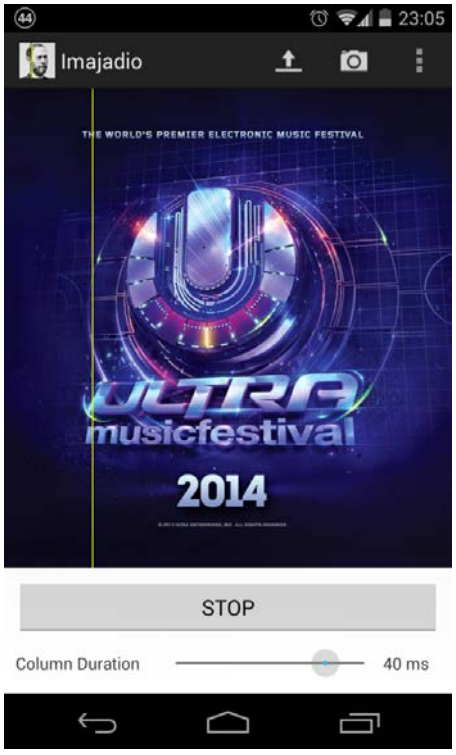
Once the user has decided on an image they can adjust the Column Duration slider. This will change the parameter that specifies how long each vertical line of pixels will be played for. The longer the Column Duration the longer the algorithm will take to complete and the length of playback. The final length of playback varies from device to device because of the difference is screen sizes and pixel density.

Press CONVERT to start the algorithm. The loading bar will take up the screen and most of the User Interface will be greyed out so there are no input ambiguities to the algorithm.



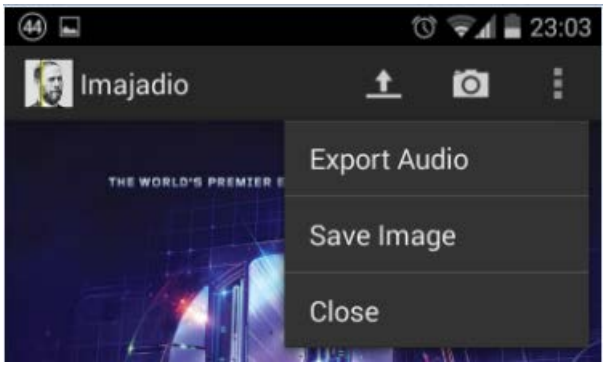
Playback Control

Actually playing the audio is quite simple once the conversion process has been completed. When done, the CONVERT button will become a PLAY button. Clicking this will begin the audio playback and start the position indicator moving across the screen. While playing, the PLAY button will turn into a STOP button which can be used to quickly stop playback and reset the audio back to the beginning.



Exporting

Exporting the picture or the audio is done very simply. These two options are found in the menu bar which can be accessed by pressing the physical menu button on your device or the three dots found at the top right of the application. You can export the picture as a .png file and/or you can choose to export the audio as a uncompressed .wav file. Both files will be stored in the Imajadio folder at the root of your device's internal storage.



Bibliography

"Audacity: Free Audio Editor and Recorder." *Audacity: Free Audio Editor and Recorder*. N.p., n.d. Web. 08 May 2014. <http://audacity.sourceforge.net/>.

Fournel, Nicolas. "AudioPaint." *AudioPaint*. N.p., n.d. Web. 08 May 2014. <http://www.nicolasfournel.com/audiopaint.htm>.

"Interference and Beats." *Interference and Beats*. The Physics Classroom, n.d. Web. 08 May 2014. <http://www.physicsclassroom.com/class/sound/Lesson-3/Interference-and-Beats>.

"Package Index." N.p., n.d. Web. 8 May 2014. <http://developer.android.com/reference/packages.html>.

Ross, B. "Discrete Fourier Transformation (DFT):." N.p., 22 Feb. 2012. Web. 8 May 2014. <http://www.cosc.brocku.ca/Offerings/4P98/lectures/DFTc.pdf>.

Wilson, Scott. "WAVE PCM Soundfile Format." *Microsoft WAVE Soundfile Format*. N.p., 2003. Web. 08 May 2014. <https://ccrma.stanford.edu/courses/422/projects/WaveFormat/>.