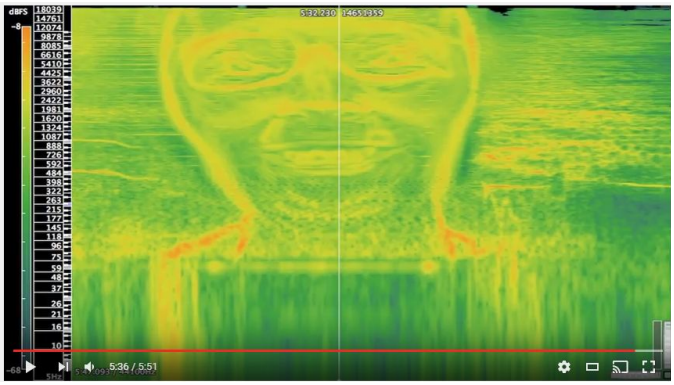
Reverse Spectrogram Synthesis

Dan Waldman & David Porter



Abstract

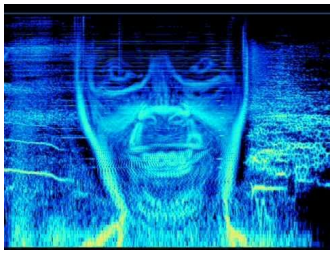
The above image is from a song created using a software program called MetaSynth that has reverse spectrogram capabilities. The song "ΔMi−1 = −αΣn=1NDi[n][Σj∈C[i]Fji[n − 1] +Fexti[n−1]]", commonly known as "[Equation]", was created in 1999 by experimental electronic artist Aphex Twin. Aphex Twin put images into the software and used them to create interesting sounds. Inspired by this creative reversal of audio visualization technology, we wanted to create our own Reverse Spectrogram using Matlab, which we knew could do basic image processing. Artists and scientists often try novel ways of visualizing music, speech and other audio. We wanted to turn that around and auralize images. We want a user to be able to input an image, say one of their face, and get out a sound that represents the image. Our specific goal: to take in an image, manipulate and analyze it, and create a sine wave representing the sum of the elements of image. We also wanted this to have creative potential, with variable parameters so a user can make adjustments to get a satisfying output.

A spectrogram is a defined as an intensity plot of the short time fourier transform (STFT). In three dimensions, they display how signal power (shown by color intensity) varies with normalized frequency (y-axis) over time (x-axis). Spectrograms are popular methods of visualizing audio because both time and frequency are essential elements of an audio signal. Our system treats the input image like a spectrogram, treating the width of the image like the time axis and the height like the frequency axis. The power comes from intensity values based on color shades in the image.

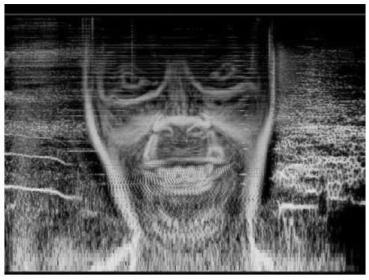
The Basic Method

At the top of the .m file, there is a section where the user can insert an image of their choice and enter desired values for the following parameters: T, the number of time slices; F, the number of frequency bins; Time, the total length of the output file; Frequency, the total range of frequencies; and threshold, an amplitude level at which a tone is omitted from the output. Our code uses these values to process the image and output audio through the following steps.

Step 1: Get image. Any .jpg image can be called into the Matlab code. This creates a 3-dimensional array representing the image, with [width X height X "RGB" color values].

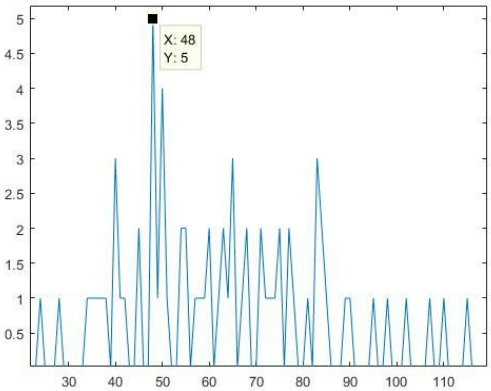
(Another version of the Aphex Twin “Equation” spectrogram)

Step 2: Convert to grayscale for simplicity. This gives us 1 value for every pixel, which we can use to get an intensity value.



Step 3: Symbolically segment the image into blocks. Here we used the user-defined values for number of time and frequency bins. We use these values to calculate the size of each time slice and frequency bin in pixels. We will run our analysis code on one block at a time.

Step 4: Get amplitude value from intensity value. Running the ‘imhist’ function on each block, we get the range of intensities for each shade on the black-to-white spectrum for that block. We then run a peak detection algorithm on the output of imhist to find the maximum value, meaning the shade with the most pixels. We take the index of that shade in the imhist array, and that value represents the shade that is most prominent in the block. We normalize these values so that they are all between 0 and 1. Near the top of the code, if the user is inclined, they can uncomment a line that will invert the shades in this grayscale image so that black is highest intensity and white is lowest intensity. This leads to different results.

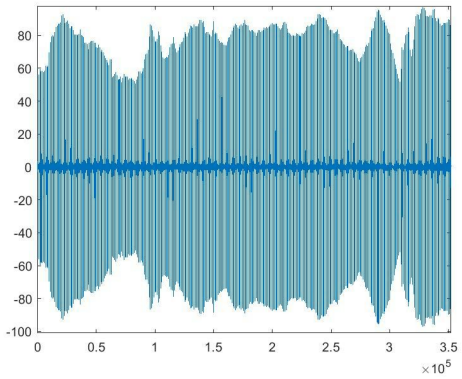


(imhist of 1 block of ‘ATE.jpg’. Value = 48)

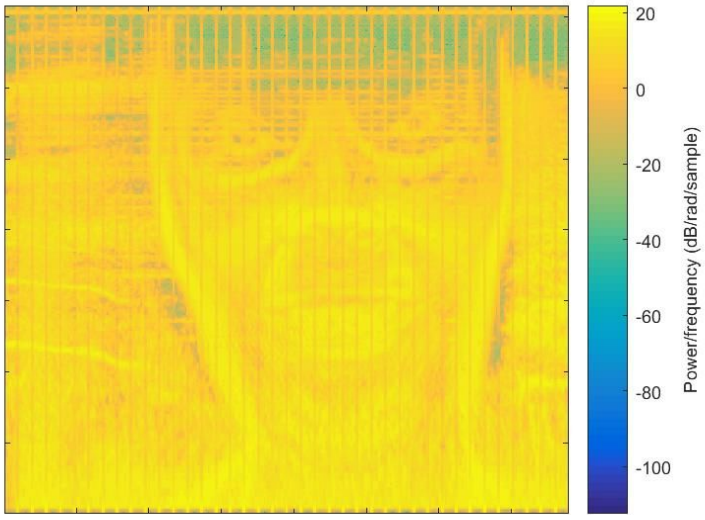
Step 5: Give frequency bins some content. Here the user-defined frequency and time ranges come into play. Based on the total frequency range, we determine the frequency range for each bin, and likewise for time. We calculate the center frequency of each bin based on the range. Now, for every time slice, we play a sine wave at the frequency of each frequency bin for the duration of the time slice and multiply it by the shade intensity value determined in Step 4. So for each block, we have a sine wave with frequency, amplitude and time.
Step 6: Sum sinusoids and output. We sum together the sine waves for every time and frequency bin. Also importantly here, we used the interpolation function 'spline' to smooth from one time slice to the next, because we were getting grating blips where the amplitude jumped. We then play the output and visualize the spectrogram of this output to verify that our system has correctly interpreted and processed the image.

Results

The output waveform of 'ATE.jpg' looks like this:



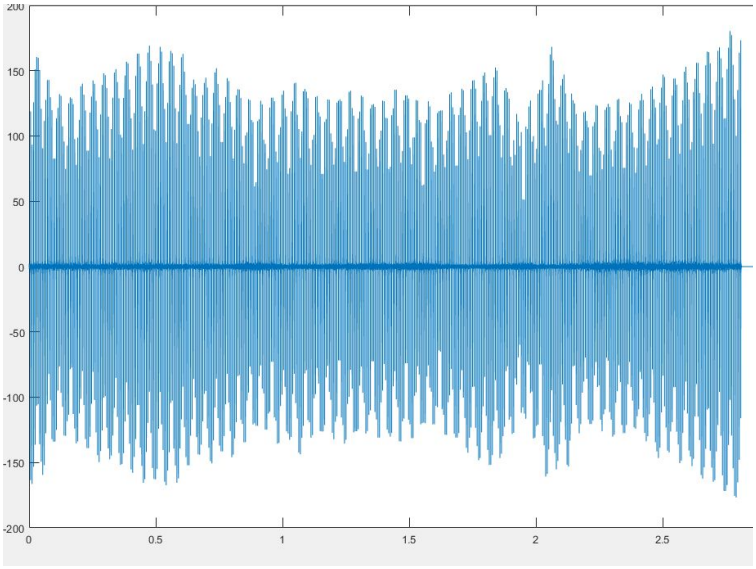
And the spectrogram looks like this:



As you can see we have returned essentially the same image. This was achieved with values T = F = 300, threshold = 0.3, Time = 8, and Frequency = 22000.
If we include the line to invert black and white in the grayscale image, we get this:



Clearly the resolution is artificially lower, and if you look at the edges of the face you can see that what was light is now darker. And the waveform:



The output sound is less complex as well, which is reflected in the waveform and the spectrogram. There is not as much variation in the frequency content over time for this particular image when black is set to be the high intensity level. For some images, however, this could be very desirable.

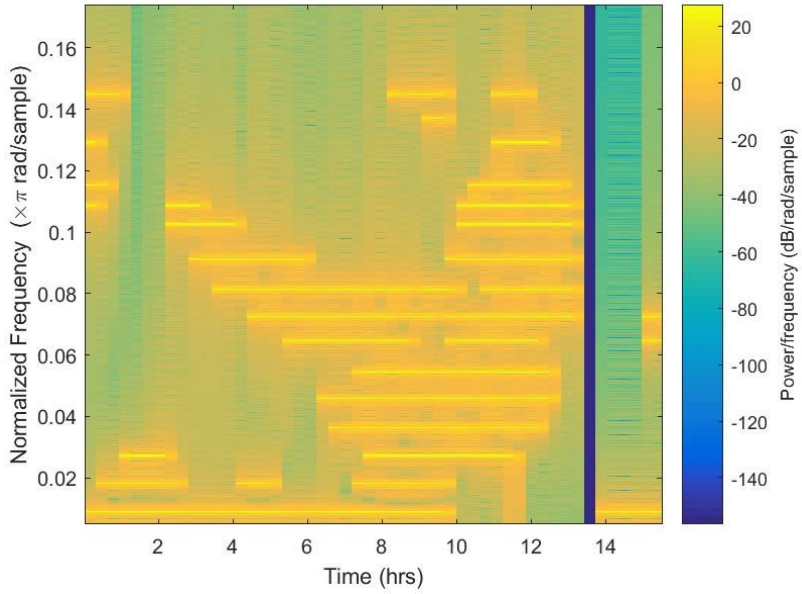
Expansions on the Concept

The primary expansion on this concept that we were interested in, which we have implemented, was to have the user be able to make the output more musically pleasing. We have done this by replacing the center frequency calculation section with calculations for specific musical notes. In the files “revspec_cmajor.m” and “revspec_harmonics.m”, there are segments of the code that create frequencies based on scales and chords, respectively. The former has an option at the top of the code for the user to choose between the C Major scale and the C Pentatonic scale. In harmonics, the notes are based on a fundamental frequency that the user can set, along with a value for F (number of frequency bins) that determines how many harmonics above that fundamental will be included -- one harmonic per frequency bin -- so that the strength of the harmonics is what varies over time. In both of these cases, we simply take the calculations

for these frequencies and put each value in a frequency bin. The next step we would take for this project is to expand the variety of scales and harmonic structures we use. There is great potential in that route for increasing the creative capabilities of this concept.



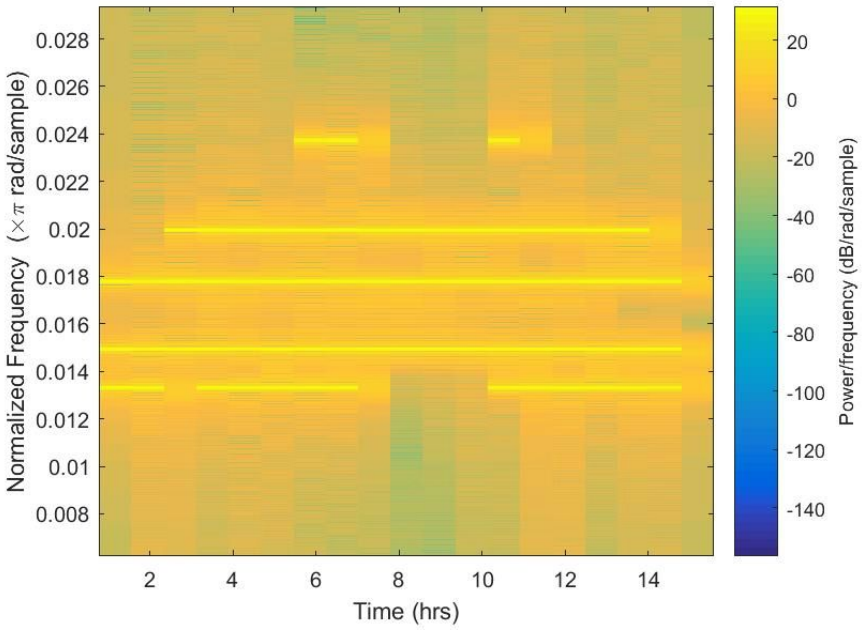
We used this picture of a dove on the harmonic algorithmic, and here is the output spectrogram:



As you can see, the resolution is somewhat lower with the more musical algorithms, however, the sound is much more pleasing. The harmonic results in a nice tone with varying harmonic structure.

The following picture of the elephant was used for the C Major scale code. This was one of our personal favorites, and the same visual and audio results hold true. Here, however, there is a noticeable melody that is quite pleasing.





Overall, this elephant was our favorite. Here is an image of the elephant put through the basic method with the center frequency calculations, shown with the spectrogram function:

