SynthBilder

MAT 201A: Media Signal Processing

August Black

1. Introduction

The transformation of an acoustic signal from the time domain into other mathematical representations of that signal is one of the key concepts of digital signal processing. Of these types of transformations, the Fourier transform is one of the most significant. The DFT (Discrete Fourier Transform) takes a discrete time domain signal and breaks it up into discrete frequency bins of magnitude and phase counterparts. This is know as the spectrum.

The Fast Fourier Transform, or FFT, is an optimized version of the DFT for computer systems. Running an FFT over an audio segment of N samples will return an array of (N/2)+1 complex numbers[1]. The real and imaginary parts of these complex numbers make up the magnitudes of cosine and sine waves with frequencies equally spaced between zero and one-half of the sampling rate. However, the addition of a cosine and sine wave at the same frequency is the same as a single cosine wave with different amplitude and shift in phase. This can be stated as such:

```
A*cos(x) + B*sin(x) = M*cos(x+ theta)
 where
M=sqrt(A^2 + B^2) and theta=arctan(B/A).
```

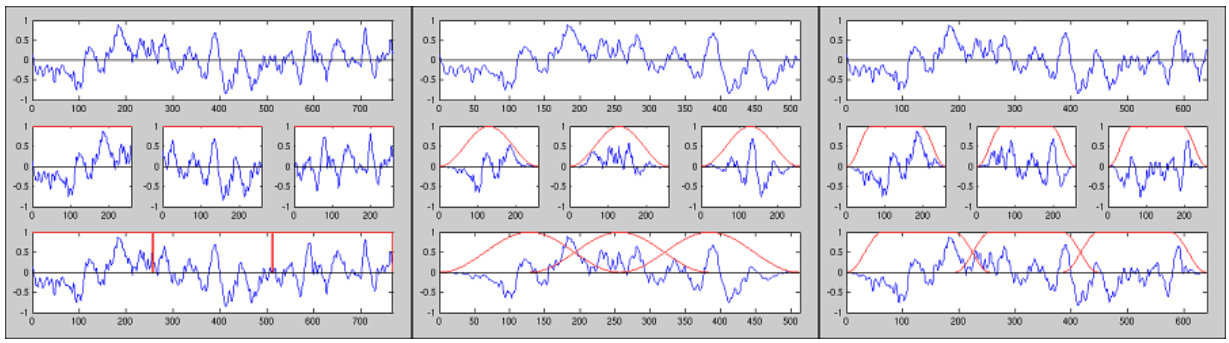
In this case, A is the real part of the array returned by the FFT and B is the imaginary part.

Inversely, to go from the spectral domain back to the time domain, one only needs to add all of the phasers from the FFT.

```
M[1] * cos( 2 * pi * f[1] * t[] + theta[1]) +
M[2] * cos( 2 * pi * f[2] * t[] + theta[2]) +
M[3] * cos( 2 * pi * f[3] * t[] + theta[3]) +
...
M[n] * cos( 2 * pi * f[n] * t[] + theta[n])
```

Another challenge for digital signal processing is windowing. The method for transforming an entire time domain signal of significant length into the spectral domain is to break the signal into equal parts (called windows), and do the FFT over each part. However, windows come in all shapes and sizes and will have a significant impact on the transformations. The simplest of these is a rectangular window. However, others such as the Hanning or Tukey window are also very useful. [2]

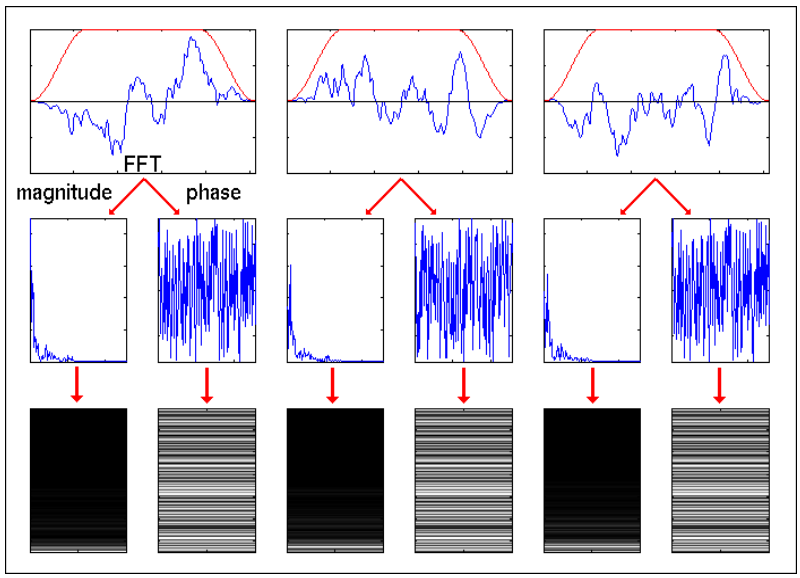
Figure 1. From left to right: Rectangular, Hanning, and Tukey windows



In the figure above you see three separate graphics. In each graphic the orignal signal is split up into 3 windows of equal length. In the middle row of each graph, you see the plot of the 3 signal segments (in blue) after being shaped by the windowing function (in red). The bottom row of each graph shows the final signal after the 3 segments have been put back together, taking overlap into account. Where the segments overlap, the signals are added. For a Hanning window, the overlap is 1/2 the size of the window. A Tukey window, however, can change shapes according to an alpha value. With an alpha of 0, a Tukey window is a rectangle. With and alpha of 1, a Tukey window is a Hanning window. It is therefore a very flexible window. For a Tukey window, the overlap is floor(window_size - alpha*window_size/2).

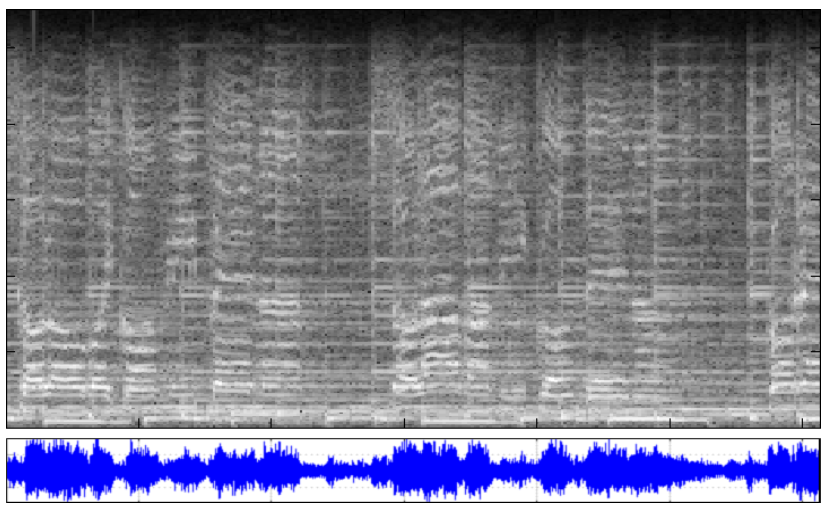
To represent an entire audio clip, FFT's are taken of each windowed segment. As said, the FFT produces an array of real and imaginary parts of complex numbers. In cartesian form, these are the magnitudes of cosine and sine waves, respectively. Conversion to polar form yields magnitude and phase of single cosine signals. In polar form, both arrays of magnitude and phase can be seen either horizontally as a plot of magnitude or phase value over frequency. Or, they can be viewed as vertical columns, or histograms, where the y axis is frequency and the pixel value (brightness) represents the magnitude or phase values. The following figure shows 3 windowed audio segments that have been transformed by an FFT. The second row shows the magnitude and phase of each segment plot horizontally (x=value,y=frequency). The third row shows a histogram of the magnitude and phase (y=frequency, x=1, z=value).

Figure 2. FFT of 3 windowed audio segments



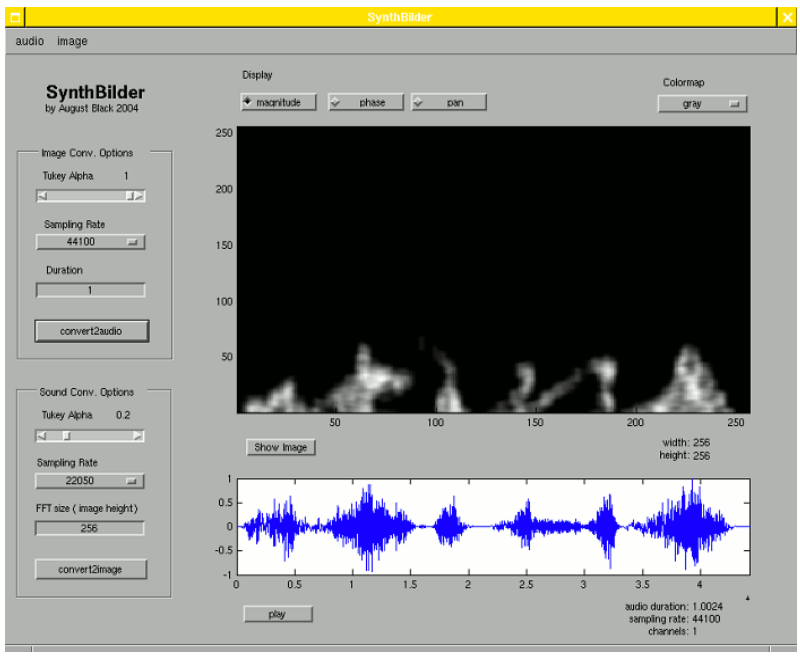
A spectrogram or sonogram is the same as above, only the single histogram columns are concatenated horizontally. Another thing to note is that a spectrogram normally only shows the magnitude portion of a signal. In many ways, a spectrogram is an image of the acoustic signal.

Figure 3. Sonogram: showing only the magnitude portion of the signals spectrum



2. SynthBilder Application

Figure 4. SynthBilder screenshot

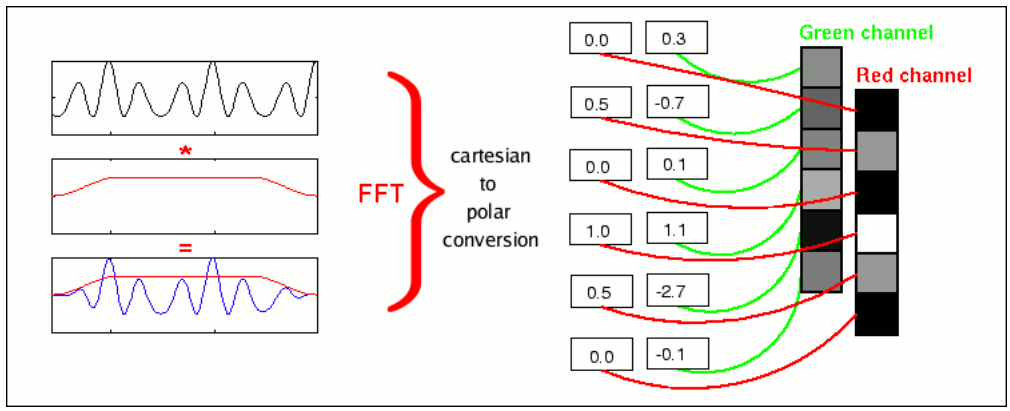


SynthBilder is a Matlab program that transforms an audio signal into a sonogram image and vice versa. It allows one to study and examine the conversion of digital audio signals from the time domain into the frequency domain and back again. Additionally, *SynthBilder* demonstrates the use of windowing in the transformation of an audio signal. But, most importantly, *SynthBilder* will allow one to compose in a visual manner with sound, and provide a fine-tuned mechanism for sound synthesis.

The first part of the Matlab application takes a sound as input and produces a spectrogram image as output. The magnitudes and phases of each windowed segment are scaled from 8-bit unsigned integers (0-255) where magnitude is mapped to the red channel and phase is put on the green channel. For now, the blue channel is unused, but is reserved for future use in describing the stereo pan of a signal. Magnitude values from an FFT range from 0 to 1, so conversion to integers means just multiplying by

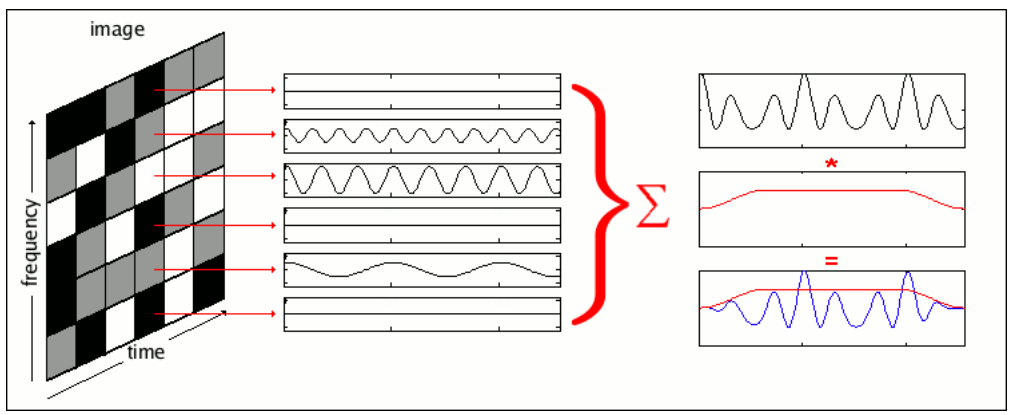
255. Since phase will range from minus PI to PI, it must first be scaled to a positive range and then multiplied by 255 as so: (phase + pi) / (2*pi) * 255

Figure 5. Sound to image conversion



The second part of the application is able to take a spectrogram as input and produce a sound. The spectrogram can be painted in any program (such as the Gimp) and loaded from a file on disk. This process is almost the exact inverse of the sound-to-image conversion, except that instead of doing an Inverse FFT, *SynthBilder* renders the sound by means of additive synthesis. The advantage of additive synthesis over the IFFT is that it allows more control for playing with time and frequencies. The following figure demonstrates how this is done.

Figure 6. Image to sound conversion: phaser addition & windowing



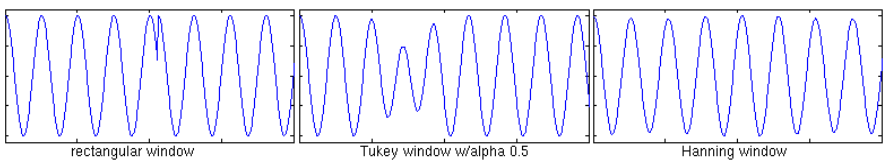
The Red and Green pixel column values of the image are converted back to magnitude and phase for specified frequencies (spaced evenly from 0 to Fs/2 in the case of *SynthBilder*) and then added and windowed. Each column segment is rendered individually, and are then concatenated with a specified overlap size to form the entire segment of audio. If the image is a single channel image, then magnitudes are used for the conversion and phase can either be set to zero overall or be randomly generated.

3. Considerations

When converting from image to sound, there is no telling if phase has been included in the image (ex. the image is black and white only). And, because *SynthBilder* should also be able to read images that don't have proper phase saved on the green channel or images that are only black and white, it is necessary to make some special considerations.

The main problem in "creating" phase for a signal is how to know the shape and direction of the tail of a window so that when the following window segment is overlapped and added to it, there is a smooth transition from one segment to the next. When the window is a rectangle, there is a big chance that the end of one segment is completely out of line with the beginning of the other. Here you notice an audible 'click'. When a Hanning or Tukey window is used, there still can be various kinds of constructive or destructive interference where the windows overlap.

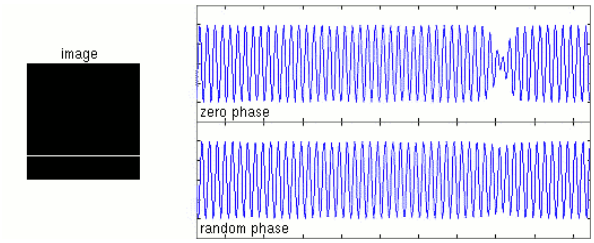
Figure 7. Phase problem seen on overlapping segments windowed in 3 different ways



The graphic above shows the transition from one signal at a fixed frequency overlapped with another signal at the same frequency. One can see how the window can smooth over the border between the two. However, a smoothing window doesn't guarantee that cancellations won't take place. When multiple signals with varying frequencies are added, the problem is expanded.

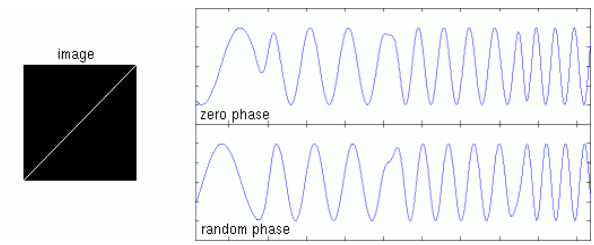
In terms of image to sound conversion, this problem can be defined as how to gracefully traverse the border of one column to the next column. The problem can best be seen when dealing with pure tones. To demonstrate this, I provide the following two graphics.

Figure 8. Phase problem on a single frequency



Both signals above were rendered with a Tukey window at alpha 0.2. The black and white image on the left represents the magnitude components of a spectrum. The two signals on the left show a close-up of the rendered audio at a place where windows have been overlapped. Above, one can see that with the same window, a change in phase can make a big difference in the signal.

Figure 9. Phase problem over changing frequency



Here, as the sound is rendered from the image, column by column (audio segment by audio segment), one should hear a single tone that is pitched from low to high (Fs/2). On the right are two plots of the rendered audio - the top one with phase set to zero and the bottom one with random phase. In both cases, the overlap from one windowed segment to the next leaves some residual of the border, however, in most cases the random phase variation sounds the most pleasing.

An idea for the future that might address this problem would be to develop a graphical synthesis technique similar to *SynthBilder* that uses vector graphics as input instead of raster images. With a vector representation of sound (as an image), there would be no column ’borders’ to traverse. That way, each sound transition "object" - a vector based line or shape - could be rendered with scalable phase based on its position in time. Additionally, it would allow for frequencies to be ramped smoothly as if it were an analogue transition from one frequency to the next.

4. Conclusion

With this project, I was able to narrow in on and understand some significant, although basic, aspects of signal processing. The next steps for *SynthBilder* are to find a clever way to include panning information for rendering to a stereo audio signal, and to perform optimizations, possibly porting the code into C for a speedier interface. Of course, the main next step for *SynthBilder* is to keep generating flavorful audio clips.

5. Related Works and References

For a online version with click-able links and color graphics please refer to either http://luv.mat.ucsb.edu/synthbilder or http://aug.ment.org/synthbilder

Loris (http://www.cerlsoundgroup.org/Loris/) - an Open Source C++ class library implementing analysis, manipulation, and synthesis of digitized sounds using the Reassigned Bandwidth-Enhanced Additive Sound Model.

Bitmaps & Waves (http://www.webcenter.ru/~vsoft/steps/bwSteps.html) - a simple program converts bitmap images to sounds, and vice versa. Every line of loaded image is assumed to be a spectrum of sound, and it is converted to sound signal by means of inverse Fourier transform

Sculptor (http://sculptor.sourceforge.net/sculptor.shtml) - is designed for the manipulation of sounds in other domains.

Metasynth (http://www.uisoftware.com/) - a fast and flexible graphic sound design and synthesis environment.

Woon Seung Yeo's MAT 310 project
(http://www.mat.ucsb.edu/~woony/research/winter01/mat310/index.html) - image sonification: image to sound.

"Equation" by Aphex Twin (http://www.visualizationsoftware.com/gram/example15.html) - Examples of Audio Spectrum Analysis

DSP Guide (http://www.dspguide.com/pdfbook.htm) - Chapter 8 and subsequent chapters of "DSP guide" on the application of the Fourier Transform.

Signal Processing First, James H. McClellan, Ronald W. Schafer, Mark A. Yoder. Pearson Education,Inc. - Chapter 3 on Spectrum Representation, especially 3.7 on Time-Frequency spectrum.

John Wilder Tukey - a short biography
(http://www.mrs.umn.edu/~sungurea/introstat/history/w98/Tukey.html)

Notes

1. Actually, an FFT will return (N + 1) complex numbers for an N-length time signal. However, N/2 of these numbers represent negative frequencies, which for many purposes can be disregarded.
2. Tukey was a prolific scientist and statistician. One of his most famous contributions to the field of information visualization is the Box-and-Whisker Plot (also available in matlab).