Michael Rawding CPEG MAJOR

ELEG 305

Computer Assignment #1

March 20, 2018

In this computer assignment, we were tasked with developing a computer program to reverberate and convolve sound signals. In the program, the convolution program was used to allocate signal direction using the impulses from the left and right channels of sound. The derivation of the difference equation and impulse response can all be seen at the end of the document. There plenty of issues I had completing this assignment. One of the longest and most annoying was that the numpy, scipy, and pylab modules were not installed in my python modules on my laptop. To combat this problem, I quickly turned to google. Initially google was no help at all, I did remote and web installations, but they were no good. I turned to command prompts, but apparently it was incompatible with my python and IDE. So after uninstalling and reinstalling Python and the IDE, I created a command window within the original python path(which also took some more googling) and learned to use the pip commands. In the command window, the statements “pip install numpy,” “pip install scipy,” and “pip install matplotlib” my program could finally build. Next came coding the my\_diff and my\_conv modules. At first it was tricky, so I went to pen and paper. After seeing the form of the convolution:

y(0) = x(0)h(0) -> y(1) = x(0)h(1) + x(1)h(0) -> y(2) = x(0)h(2) + x(1)h(1) + x(2)h(0)

I wanted to be able to shift and multiply the vectors and append them to the output without using a for loop. However, since my python skills were rusty a settled for a simple iterative for loop. Results show that the myconv function is extremely slow because of nested for loops that iterates through all shifts of the impulse until there is no longer a overlapping points in the arrays. This function takes a very long time, and would not recommend use in most applications, however after awhile it gets the job done. The next part of the coding was finding the difference equation for the output signal of a sound with reverberation. I decided to break up the sound signal into three parts: Original signal(no echo), Original signal(no echo) + the echoes from previous x[n], and finally when x[n] ceases, but the echoes and reverb remain. So we know that originally we know that the output y[n] = x[n], just a clean sound signal. Then once the echoes hit, we know y[n] = x[n] + (decay factor)\*y[n-N], and lastly when the x[n] signal ceases, but the reverbs remain we get y[n] = (decay factor)\*y[N-n]. So using for loops again, the loops iteratev over n for each of the three “components” of the output signal. In this reverberation model, the decay = alpha, and the delay = N. since 0 < alpha < 1, the signal amplitude “decays” because the reverb y[n-N] is scaled by alpha for every y[n] that occurred before. The N is the delay factor. This means that it is the delay from where the original ouput y[n] = x[n] is then reverberated and paired with the next output. i.e N = 1 and alpha = 0.5: y[0] = x[0] -> y[1] = x[1] + 0.5 \* y[1-1] -> y[1] = x[0] + 0.5\*x[0].

\*it takes 1 discrete time increase of n for the original signal to return and add as a scaled version of the previous input\*\*

It shows above that the N delay is the amount of n discrete samples of x[n] that a clean signal y[n] = x[n] until n = N, and the scaled version of previous y[n] returns with the output and turns into y[n] = x[n] + a\*y[N-n].

A more complicated model of reverberation is shown at the end of the doc, but it involves where the direct component and the early echoes with an FIR filter and the reverberation tail with an Infinite Impulse Response (IIR) filter. It is shown below.