COS/MUS 314

Assignment 5: Fourier Analysis

Assignment due 9 April 2012, 11:59 pm

This assignment is done with your partner.

Reading & Resources

ChucK complex and polar data types:

http://chuck.cs.princeton.edu/doc/language/type.html#complex

ChucK FFT documentation: *Read about Unit Analyzers, especially the FFT datatype and its functions.*

http://chuck.cs.princeton.edu/doc/language/uana.html

FFT examples:

http://chuck.cs.princeton.edu/doc/examples/analysis/fft.ck
http://chuck.cs.princeton.edu/doc/examples/analysis/fft2.ck

FFT handout: *We haven't talked about all of this in class yet, but you may find it useful.*

http://www.cs.princeton.edu/~fiebrink/314/2010/week8/FFT_handout_2010.pdf

Optional further reading on Fourier analysis and FFT:

http://music.columbia.edu/cmc/MusicAndComputers/chapter3/03_03.php
http://music.columbia.edu/cmc/MusicAndComputers/chapter3/03_04.php

ChucK's FFT Unit Analyzer in a nutshell

ChucK's FFT object allows you to compute the spectrum of a sound over a window of time. To take an FFT, you specifically need to do the following:

1. Instantiate an FFT object. For example:

```
FFT f;
```

2. Connect this object to your sound source using => . Also connect the object to blackhole (because blackhole will "suck" the samples from your FFT, which will in turn pull samples from your sound source… no blackhole means no sound to analyze). Note that, just with Unit Generators, you can instantiate the object on the same line as connecting it within your patch. For example, to take an FFT of sound coming in from the microphone:

```
adc => FFT f => blackhole;
```

3. Set the parameters of the FFT. Specifically, you need to set the size—that is, how many samples of sound are in your analysis window? Equivalently, how many "bins" will your FFT have? In class, we referred to this as "N". N should be a power of 2 if your FFT is to truly be "fast". Common sizes for this type of analysis are 64,128, 264, 512, 1024, 2048, 4096.

You should also choose a window function and its size. We haven't talked about windows in class yet, but you can get a glimpse of how this works by looking at the FFT handout (link above). For now, it's fine to use a Hamming window whose size is either equal to the FFT size or ½ of the FFT size. For example, this code gives you a 1024-point FFT with a 512-sample Hamming window:

```
1024 => f.size;
Windowing.hamming(512) => f.window;
```

4. Let some time pass. This will fill up the FFT object's internal buffer of sound samples. (The FFT maintains a memory of the last "N" audio samples generated by the sound source attached to it.) For example:

```
.2::second => now;
```

5. Trigger the FFT computation using the .upchuck() function. This causes the object to compute the FFT on the last N samples (which are by now in its buffer).

```
f.upchuck();
```

6. Access the results of the FFT by using the .fval(i), .fvals(), .cval(i), or .cvals() functions. .fval(i) returns the **magnitude** of the i-th FFT bin, and .cval(i) returns the **complex value** of the i-th FFT bin. .fvals() returns a float array of all N magnitude values, and .cvals() returns a complex array of all N complex values.

Recall that the FFT divides up the frequencies between 0 and the sampling frequency into N evenly-spaced "bins." Specifically, the i-th FFT "bin" corresponds to the frequency F_{SR}*(i/N) where F_{SR} is the sampling frequency and N is the FFT size. As a simple example, if you take a 4-point FFT (N = 4) and your sample rate is 44,100 Hz, then bin 0 will correspond to a frequency of 44,100*0/4 = 0Hz, bin 1 will correspond to a frequency of 44,100*1/4 = 11,025Hz, bin 2 will correspond to a frequency of 44,100*2/4 = 22,050 Hz, and bin 3 will correspond to a frequency of 33075Hz.

Also remember that we only need to look at the first N/2 bins, since we're dealing with a digital signal and, as a result, any frequency above $F_{SR}/2$ (the Nyquist frequency) will be indistinguishable from a frequency below $F_{SR}/2$ (its alias). So in the example 4-point FFT, we only care about bins 0 and 1.

Example to print out the magnitude and then the complex value of bin 0 (after upchuck has been called):

```
<<< f.fval(0) >>>;
<<< f.cval(0) >>>;
```

7. Repeat steps 4-6 ad nauseam in order to capture a sequence of FFTs describing the spectrum of a sound as the sound changes over time.

Question 1. FFT practice (6 points)

a) An N-point FFT (i.e., an FFT of a window of N consecutive samples in time) will yield N frequency “bins”, equally spaced over the range of 0Hz (“DC”) to the sampling frequency (e.g., 44.1 kHz).

Knowing this, discuss one possible advantage and one possible drawback of choosing a large value for N.

b) Download the skeleton code from
http://www.cs.princeton.edu/courses/archive/spring12/cos314/assignments/assignment5/Assignment5Skeleton.ck

Currently, this code computes the FFT of the most recent N samples (where N=1024), and repeats this every 0.1 seconds.

Fill in the code skeleton to locate the index of the frequency bin with the highest magnitude. Use your knowledge of the FFT and the sample rate used by miniAudicle (not sure? Go to Preferences menu) to compute the frequency (in Hz) associated with that bin. Print out both the index and the associated frequency.

There are examples of this sort of simple ChucK pitch tracker online. **Do not seek them out or look at them.** The code for this question must be entirely your own original work.

Congratulations! You now have a simple pitch tracker. ☺

c) Plug in headphones (and restart the virtual machine). Now, within the same ChucK file, create a SinOsc and continually update its frequency to be the frequency found by your pitch tracker. (Now you can hear the pitch identified by your pitch tracker instead of trying to guess whether the values it prints out are correct.)

d) Experiment with different input signals and describe what you find. Sing or play an instrument into the microphone. Does the FFT peak frequency correspond to the true fundamental? If not, do you see any pattern in the mistakes it makes? Also experiment with speech, noise, and other sounds. Do you notice anything interesting?

Question 2: Extra FFT practice (4 Points)
Pick (at least) one of the following options to get some more practice with FFTs.

a) Pitch tracker improvement: Describe a few ways that you might improve your pitch tracker to be more robust, accurate, useful, etc. Implement at least one improvement. Submit your updated code and your written comments on why you did what you did and whether you succeeded in improving performance.

OR

b) Additive synthesis from FFT: Analyze a recording of some pitched sound (e.g., a voice, instrument, bell, etc.) using Audacity's "Plot Spectrum" tool. (Don't use the spectrogram – just look at an FFT at one point in time, by selecting a section of audio with the mouse and clicking on Analyze→Plot Spectrum in the menu bar.) Identify the most prominent frequency components by finding peaks in the spectrum. Play around with the FFT size and window type to try to get the most useful peaks. (Notice that the tool will "snap" the vertical cursor to peaks and display the precise frequency value at the peak, right below the spectrum.) Also pay attention to the relative strengths of the peaks. Audacity gives you peak values in decibels and not absolute amplitudes, so you should know that a change of 10 dB is equivalent to approximately a factor of 3.16 in amplitude.

Use this knowledge of peak location and relative strength to resynthesize the sound using additive synthesis (i.e., with a finite number of SinOscs with specific frequencies and amplitudes).

Comment on the results. Is your synthesized sound a good match to the original? How sensitive is the quality of your synthesized sound to the number of SinOscs you use? How might you further improve the quality of the synthesized sound?

Submit your synthesis code, your recorded sound example, and (if necessary) information about the precise time point within the sound example that you are trying to resynthesize.

OR

c) Implement a phone dialing interpreter in ChucK:

When dialing, landline phones represent each dialed digit using two simultaneous sine waves at specific frequencies. Take a glance at http://en.wikipedia.org/wiki/Dual-tone_multi-frequency to learn more. Note the DTMF frequency table, which you could use to synthesize perfect dialing signals in ChucK using only two SinOscs!

Now, implement a phone dialing interpreter. In the simplest case, your interpreter will be a piece of ChucK code that listens to audio input consisting of a single dialed digit (with no silence, noise, etc.) and accurately identifies (and prints out) the digit. If you want to do better, make a dialing interpreter that can accurately handle silence (i.e., no digit being dialed) and sequences of consecutive keys, so that you can play it an entire phone number and it will print out the correct number sequence. Test your code using a real phone or a simulator such as http://sio.midco.net/~dfranklin/phonedial/.

Submit your code along with a short statement describing what you did and how well you believe it works. If you use an FFT in your analysis, defend your choice of FFT size.

What to turn in:

- Written work for questions 1a, 1d
- Code for questions 1b and 1c (could be the same piece of code)
- Materials for whichever part of question 2 you chose. If you submit answers to more than one part of question 2, we will give you feedback on everything and we'll assign you the grade of whatever part gives you the highest score.