Guide to BOOK Chapter 7 Examples:
Victor Lazzarini

Spectral Audio Programming Basics – The DFT, FFT and Convolution

These examples will build on most systems, and have been tested on OS X, Linux and Windows. To build the examples you will first need a C compiler. I recommend using **gcc**, if you have a choice (on OSX and Linux you'll get it by default). On Windows **gcc** can be got by either installing **MinGW/MSYS** or **Cygwin** (www.cygwin.com).

Examples are built using **scons**, which is a handy utility for code maintenance and programming projects. It looks after re-building any files that have been modified and gets all the right components together to build the whole project. See the 'scons-guide' document for more details. If you don't have **scons,** you can get it from www.scons.org, and you will also need Python, which you can get from www.python.org. Both software are of easy installation.

You will also need to install **libsndfile**, if you do not have it already. In fact, **scons** will look for it and tell you if you don't have it. Then you can get it from www.mega-nerd.com/libsndfile**.** It's all very simple to build (if you need to) and install, all the instructions for your system are provided.

Once your system is ready for it, all you need to do is type the following at your shell/terminal ($ is the prompt) and all examples will be built.

```
$ scons
```

Because the example programs for this chapter require several source files, building them from the command line directly with **gcc** is very tricky and is not recommended.

dft

This is an example of an application of the Discrete Fourier Transform. It takes in a soundfile and converts it to its spectral representation printing out the magnitude values of each bin (positive frequencies only). It also writes a text file called *spec.txt* containing, in the first line, the input file sampling-rate, and then on each subsequent line, a bin magnitude (also only positive frequencies). The main program is in *dft_main.cpp* and the DFT operation itself is found in *dft.cpp*. The DFT is taken of the whole soundfile, so if you use a large soundfile, it will be very slow. Try first with soundfiles the provided here. See **Project 1** for some ideas on how to modify the program and give it more flexibility.

spec.py

This is an example of how the DFT output data (magnitudes) can be displayed. It invokes the **dft** program above to read a soundfile and displays a *magnitude* x *frequency* graph. In order to use it, you can type

```
$ python spec.py soundfile
```

convolution

This program implements the basic application of a straight DFT analysis (using the FFT algorithm), fast convolution. It takes in one soundfile (the impulse response) and multiplies its spectrum with the spectrum of another input file, realizing the equivalent operation to time-domain convolution. Try different types of impulse responses (which can be anything, from a room impulse response to another soundfile that you want to cross-synthesize with your input). The main program is found in the file *convolution_main.cpp* and the convolution operation itself in *convol.cpp*. **Project 2** has some ideas on how to enhance this program.

Projects

Here I'll propose some ideas that you can try for yourself. These will be mainly based on enhancing the examples that have been provided for this chapter. However, they'll give you some useful insight into audio programming.

Project 1

This project is divided into two steps, which can be done separately. The first one is to modify the **dft** program so that instead of loading all the samples off a soundfile and then processing them in one go with the DFT, you divide it up in blocks of *N* samples and then process them in a sequence. In that case you will then output blocks of *N/2+1* magnitudes (positive spectrum only). This will allow you to see how the spectrum of a sound changes in time. There is only one major modification to be made to the *dft_main.c* function: you will need to repeatedly read blocks of audio from the file, call the dft() function and printout the results. A small hint on how to achieve this: add an inner loop that will do that for each block of audio.

Once the **dft** program has been enhanced, the second step is to see if you can then animate the spectral display in *spec.py.* This is the second step and will require you to know a bit about Python programming (which is very easy, if you know C++). The graphical interface was done using *TkInter*, which allows for easy programming. In www.python.org you will find plenty of documentation on how to do all of this. You can perhaps add a slider that can make the graph move from one analysis block to another, or perhaps a button that will trigger the animation.

The ideas in this project are very much linked to the principles discussed in the chapter "The STFT and Spectral Processing". They will provide a nice practical introduction to them.

Project 2

The convolution program in *convol_main.c* allows for a mono impulse response to be applied to a mono soundfile. A more complete version of it would take in two stereo files and process them. To do this, you will need to process the channels separately and then output them as a stereo file. Stereo soundfiles in the most common formats will be stored in interleaved fashion, ie. a sample of channel 1 is followed by a sample of channel 2 and so on. Each group of samples is called a 'frame' and libsndfile provides functions to read/write frames instead of single samples off a soundfile. Check the libsndfile documentation for it (you can do it online at www.mega-nerd.com/libsndfile if you don't have it in your computer). Once you have done this, you can try generating stereo impulse responses (or getting them from online sources) to apply to your sound. A simple way of getting the impulse response off a room is to play a very short noise burst into it, recording the result. Then edit out the noise burst and you'll be left with something very close to the impulse response of the room.