Guide to BOOK Chapter 8 Examples:
Victor Lazzarini

The STFT and Spectral Processing

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 OS X 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.

stft

This example demonstrates the use of the STFT and ISTFT operations and how they are transparent (ie. the inverse operation recuperates the original signal). It also serves as the basis for some of the exercises/projects in this chapter, as the processing operations can be applied for the spectral data that is generated in between the two transforms. The main function is found in *stft_main.cpp* and the *stft()* and *istft()* operations in *stft.cpp*. If you want to use these functions in your projects, you will need to add that file to your project build (in fact all further examples for this chapter will also include it).

simpflt

This program shows the simple LP filter described in the chapter. It takes an input and applies the filtering. Since this filter has a very gentle slope, you will need to find some sounds with high-frequency content so that the LP response is perceived.

specreson

This program implements a varying-frequency spectral resonator. The centre frequency for the filter sweeps from lo to hi and the effect is quite noticeable with any input. **Project 1** proposes some ideas of how to add more flexibility to this program.

specomb

This example demonstrates another filter, now a comb filter, which can be used for flanging effects. It sweeps the spectrum providing the typical whooshing effect that is normally implemented with a variable delay-line and feedback. See **Project 2** for some enhancements to this program.

cross

Cross is a vocoder-like program that allows for cross-synthesis of the magnitudes and phases of two soundfiles. The first soundfile provides the magnitudes and the second the phases, resulting in a sound that will have the characteristics of both. For best results, use a sound with a very interesting spectral envelope (such as a voice) for the magnitude input and a sound rich in components (such as buzzer) for the phase input. See **Project 3** for ideas on how to make this basic program more interesting.

Projects

Here I'll propose some ideas for you to try it 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

The variable filter implemented in **specreson** has a fixed center-frequency envelope and a fixed bandwidth. You can create a more flexible program by letting the user specify the envelope parameters for frequency and bandwidth. You can try first implementing a 3-stage envelope for both, taking the arguments from the command-line. In a further improvement, you can let the user specify the envelopes in two text files. These envelopes then can have any number of segments, specified in breakpoint format: time followed by value. Hint: each section of the envelope is a line and can be implemented using a linear function $y = ax + b,$ with b = start and a = (end – start)/duration.

Project 2

The spectral comb filter implemented in **specomb** also only implements a simple one-speed flanging. Here are some ideas for improving it:

1. Provide user control for modulation rate and depth.
2. Create a stereo flanger: ie. the mono input is modulated independently in two channels. You will need to double the comb processing and inverse transform, then write a stereo output (remembering that stereo files are interleaved)
3. Add an envelope for rate and depth (ie. make them time-variable). Remember that if the frequency varies, then it has to be defined in terms of phase increments (see Book Chapter 5 – *Introduction to Digital Audio Signals* for more details.)

Project 3

The cross-synthesis operation as implemented in the **cross** example has no control of nuance. An enhancement to that program would be to allow for different levels of cross-synthesis to occur, from nothing to full effect. To do this you could:

1. Take phases of input 2 and magnitudes of both inputs. Then interpolate between the two magnitudes using a depth control, say 0 for magnitudes from input 2 (no effect) and 1 magnitudes from input 1 (cross-synthesis):
    a. mag_output = (1 - depth) * mag1 + depth * mag2
2. Use an envelope to change depth in time. Allow the user to define the envelope shape.
3. You can try doing the reverse, taking magnitude of input 1 and interpolating between the phases of inputs 1 and 2. It's worth a try, but you will see that the results will be much worse in the in-between stages (phases cannot be simply interpolated). This in fact is one of the advantages of the Phase Vocoder over simple STFT, which you will be able to see in the Book Chapter 9 – *Programming the Phase Vocoder*.