Guide to DVD Chapter 2 Examples: *Richard Dobson*

Digital Filters

Building the Examples

This guide assumes you are familiar with *make* and the general build process. (See the guide document for Book Chapter 2 for a detailed explanation of these.) If you have built and stored *libportsf.a* in a single standard location, modify the makefiles accordingly.

gcc (OS X, Linux, MinGW)

Type `make` in the examples directory. This will build the *portsf* library and the four primary filter demonstration programs *biquad*, *filtsf*, *bwfilter* and *rbjfilter*. It will also build the direct convolution demonstration program *conv* in its own directory. You will need to `cd` to that directory to try the program out.

Note that the *portsf* directory contains two test programs which demonstrate/exercise the file seeking and overwrite functions: *sfreverse* and *sfrewind*. These are standard test programs for *portsf* and not relevant to this chapter.

Windows Users

It is recommended to use the *MinGW* environment for building programs on Windows. (See the guide document for Book Chapter 2 for details.)

The Programs

These programs are written as demonstrators of filter implementations. With the possible exception of *rbjfilter*, they are limited from a creative point of view. In particular, they will only process a mono soundfile. Automatic control of gain is kept to a minimum. With the exception of *biquad*, the usage for the filter programs is almost identical – the command lines are very similar. The programs differ only in the internal details of the filter implementations. In consequence, there is a degree of duplication of function – standard filter types such as the band pass are implemented in all the programs.

The filters should be explored in tandem with the chapter text and the display program *FilterResponse* (see DVD Chapter 23). The latter allows coefficient values to be entered and freely modified. In all cases, pay attention to the reported PEAK level. Sometimes this may be very low; but if the peak is reported as 1.0 or above, it may be unwise to audition the file (if at all) at other than low monitoring levels. It is recommended to use a display program such as *Audacity* to inspect such output files before rendering.

OS X Users: To open a soundfile in *Audacity* from the command-line, us the -a flag to indicate the application to use:

```
open -a audacity.app  coolsound.wav
```

Each program (with the exception of *biquad*) writes the computed coefficients for the filter to the screen (in the case of time-varying parameters, based on the first value). These can be copied into *FilterResponse* for analysis. It is worthwhile to compare the coefficient values from different programs for the same input arguments. Despite being computed in very different ways, these coefficients often end up very similar. This observation forms important background to the *Filter Toolkit* written by Robert Bristow-Johnson (see DVD Chapter2).

Impulse Testing

All the filter programs will take the special filename *impulse* as an alternative to a soundfile. The programs will then apply a single impulse to the filter and output a short 4096-sample soundfile comprising the filter impulse response. The properties (44.1KHz, floats) are hard-wired in the code and can be changed if desired. The impulse response file can be loaded into Audacity – not to play, but to analyze. Select the whole file and activate the spectrum analysis facility. The FFT display shows the frequency response of the filter. Note that you need to use the *rectangular* window option.

Two source files are provided:

pulse44-200.wav : pulse wave 100Hz with 200 partials
pulse44-50.wav : pulse wave 100Hz with 50 partials

The examples below all use the second file. The two are identical apart from the number of partials. This has consequences with respect to output level. Simply put, with 200 partials the strength of each one is somewhat less than that of a partial in the second file. A bandpass filter, in particular, will remove proportionately more of the energy of the first file, resulting in significantly reduced output level. The reader is encouraged to use the resources previously developed (*oscgen*, *tabgen*, etc.) to generate their own collection of test sources.

biquad – Apply raw biquad coefficients to a soundfile.

Usage:

```
BIQUAD infile outfile a0 a1 a2 b1 b2
```

As its name suggests, this is the most low-level filter program possible. It takes five filter coefficients (unused ones should be entered as zero) that are applied to the input file. Should you discover what appears to be a viable filter using *FilterResponse* (try to choose one that will not "blow up" or generate over range values!), use this program to test it.

Without description, the coefficients below are taken from *Computer Music* by Dodge and Jerse[1] (see DVD Chapter 2 for more details); use *biquad* (and *FilterResponse*) to determine their characteristics.

a0	a1	a2	b1	b2
0.14509	0.0	0.0	-0.85491	0.0
0.03263	0.0	0.0	-1.79764	0.0
0.03780	0.0	-0.03780	-1.7779	0.92439
0.86327	-0.66072	0.86327	-0.66072	0.72654

filtsf – Explore filter designs from *Computer Music.*

Usage:

```
FILTSF infile outfile type cfreq [Q]

      filter types:
          0 = LowPass
          1 = HighPass
          2 = allpole bandpass
          3 = polezero bandpass

     cfreq  = centre/cutoff freq (0 < cfreq < Nyquist)
              cfreq may also be a breakpoint file

     Q      = resonance factor (1 - 100)

     Q is required for bandpass options.
```

See the text for descriptions of the four filter types. Note especially type 3, which implements the "improved reson" filter.

Example: Apply a low pass filter with reducing `cfreq` to pulse44-50.wav.

Create the breakpoint file *cfreqdown.txt* with this data:

```
0.0 2000
5.0 50
```

```
./filtsf pulse44-50.wav pulfilt0.wav 0 cfreqdown.txt
```

Example: Use the polezero bandpass with Q = 10:

```
./filtsf pulse44-50.wav pulfilt3.wav 3 cfreqdown.txt 10
```

bwfilter – Explore Butterworth filters from *Computer Music.*

Usage:

```
BWFILTER infile outfile type cfreq [Q]

filter types: 0 = LowPass
              1 = HighPass
              2 = BandPass
              3 = Notch

  cfreq    = center/cutoff freq (0 - Nyquist)

  Q        = quality factor for bandpass and notch filters
               (0 < Q <= 100)
             filter bandwidth set as cfreq / Q
             (Q is required only for types 2 and 3)
```

See the DVD Chapter 2 for more on the significance of the Butterworth filter.

Again this is a minimal implementation. Apart from the one change of filter type, the command line is unchanged from *filtsf*.

Example: A descending notch filter with low Q.

```
./bwfilter pulse44-50.wav pulbw3.wav 3 cfreqdown.txt 2
```

The effect of a notch filter is especially subtle; typically employed to attenuate some unwanted part of the spectrum of a sound. The use of standard test waveforms is perhaps less appropriate here, though in this case the effect should still be clearly audible.

rbjfilter – Explore the filters described in the *Filter Toolkit* by Robert Bristow-Johnson. The documentation covered in DVD Chapter 2 should ideally be read before exploring this program.

Usage:

```
rbjfilter infile outfile type cfreq Q  gain [dBGAIN]

filter types: 0 = LowPass
              1 = HighPass
              2 = BandPass
              3 = Notch
              4 = Allpass
              5 = low shelf  (requires dBGain)
              6 = high shelf (requires dBGain)
              7 = peaking EQ (requires dBGain)

cfreq           = center/cutoff freq (0 - Nyquist)
```

```
Q               = resonance factor   (0 < Q <= 100)

gain            = gain factor applied to output. gain > 0.0.

dBGain          = cut/boost (dB) for shelf and peaking EQ filters
```

Examples: Test low and high shelf filters.

```
./rbjfilter pulse44-50.wav pulrbj5.wav 5 1000 4 1 12
./rbjfilter pulse44-50.wav pulrbj6.wav 6 1000 4 0.25 12
```

Note that the second example requires gain scaling. It is important to investigate the interactions of `Q`, `gain` and `dbGain` in these and similar cases. A moderately high value of `Q` is used here deliberately to demonstrate the peaking effect even of a shelf filter – in the end these are all "mere" biquad filters, and it is all too easy to expect more of them than they are capable.

Note that the allpass filter, by definition, has a flat magnitude response. Allpass filters are best explored (a) by mixing the input with the output, to obtain phase cancellation effects and (b) by using time-varying `cfreq`. Many classic audio effects such as the phaser can be implemented using a cascade (series connection) of allpass filters with sinusoidal or random modulation of each cutoff frequency.

Notes

[1] Dodge, C. and T. Jerse 1985, *Computer Music.* New York, Schirmer Books.