Guide to DVD Chapter 4 Examples: *Richard Dobson*

Audio Processing in C++

The example programs associated with this chapter fall into two groups. The first comprises C++ versions of the C programs ***siggen*** and ***vdelay*** presented in DVD Chapter 1. The C++ versions are called ***siggenpp*** and ***vdelaypp***, each in their own subdirectory. The functionality of *vdelaypp* is unchanged from the original, thus the same usage applies; but *siggenpp* supports an additional band-limited pulse wave option.

The second group comprises a number of very small test and example programs that illustrate or explore various aspects of C++, some simple, others less so. They neither process nor generate audio. Some are mentioned in the chapter text either by name or by code example, several others are not; the reader is asked to explore them as if for the first time. None of the programs deals with aspects not mentioned in the text; however in a few cases the mention is fleeting (e.g. in one or other of the endnotes). In several cases the comments in the code suggest possible lines of exploration, and even exercises. For this reason the descriptions given here are more brief than usual.

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)

For *vdelaypp* and *siggenpp* type `make` in the examples directory. This will build the *portsf* library as required. The executables are copied to the examples folder ready for use. Note that the *portsf* directory contains two test programs which demonstrate/exercise the file seeking and overwrite functions: *sfreverse* and *sfrewind*. See the file *readme.txt* in the *portsf* directory for more information.

For the test programs `cd` into the tests directory and type `make`.

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. Note that the programs will either not build, or

behave incorrectly, if built with older compilers such as Microsoft Visual Studio 6. In particular, in that compiler `new` does not throw an exception, and there are other anomalies. Readers preferring a graphic environment will need to install a later version of Visual Studio such as one of the more recent 'Express' versions.

The Programs

One source soundfile is provided: *fluteC3.wav*

WARNING: The *vdelaypp* program allows the use of feedback. This can very easily result in overloads (amplitudes increasing without limit, but clamped to digital peak). Note that the program displays the PEAK level – if this indicates 1.0, it almost always means the output has overloaded. Avoid playing back such files! They are best loaded into a waveform editor such as Audacity for viewing, and playing if it is appropriate to do so.

vdelaypp – apply time-varying delay to input soundfile.

Usage:

```
VDELAYPP [-gDGAIN][-rFREQ] infile outfile delay freq mod
                feedback wet/dry tailtime

-gDGAIN :  scales feedback signal level by DGAIN. Must be > 0.0.
         may be needed with high feedback, to keep signal within range.

-rFREQ  : vary modulation depth randomly at rate FREQ Hz.
                  0.0 < FREQ <= 100.0

delay   :  sets primary delay time in msecs. Must be > 0.0.
               Overall maximum delaytime is extended by delay * mod.

freq    :  modulation frequency (Hz). Must be > 0.0.

mod     :  percent deviation relative to delaytime. 0 <= mod <= 1.0.

feedback:  set feedback level. 0 <= feedback <= 1.0.

wet/dry :  set level of wet (delayed) signal. 0.0 < wet <= 1.0.
           The dry (source) level is set to 1.0 - wet.

tailtime:  extra time (secs) added to infile to accommodate decay tail.
```

Example: process the file *fluteC3.wav* to create wide slow random frequency variations.

```
./vdelaypp -r2 fluteC3.wav flutevary.wav 200 1 0.5 0.5 1 2
```

Example: apply vibrato (frequency modulation). Uses sine modulation, no feedback.

```
./vdelaypp fluteC3.wav flutemod.wav 200 4.5 0.005 0 1 1
```

Example: as above, with random variation of depth.

```
./vdelaypp -r2 fluteC3.wav flutemodr.wav 200 4.5 0.008 0 1 1
```

siggenpp – generate simple waveforms.

Usage:

```
siggenpp [-sN] outfile wavetype dur srate nchans amp freq

where wavetype = :
                 0: sine
                 1: triangle
                 2: square
                 3: sawtooth up
                 4: sawtooth down
                 5: band-limited pulse

-sN: set sample type to one of:
                 0: 16-bit (default)
                 1: 24-bit
                 2: 32-bit integer
                 3: 32-bit float
```

Example: generate stereo pulse wave.

```
      ./siggenpp blpulse.wav 5 5 44100 2 0.5 440
```

Test Programs

Programs provided: ***buftest, constcast, defargs, delaytest, dyncast, etscale, exception, memtestdjd, runerr, sfx, tempfib, testcomplex, testcon, testmin, tststrm, testtbuf***

None of the programs takes any command line arguments.

Note that two of the programs, *buftest* and *delaytest*, access source files in the *vdelaycpp* directory.

Summary of the Programs

buftest – exercises the buffer class in *vdelay.h/.cpp.*

constcast – demonstrates the use of `const_cast`.

defargs – demonstrates use of default arguments to a function.

delaytest – exercises delay class in *delay.h/.cpp* and demonstrates output text formatting.

dyncast – demonstrates `dynamic_cast` and `typeid`, exception handling.

etscale – simple class; static variables; compute arbitrary equal-tempered intervals.

exception – simple demo of custom exception; trapping new failure.

memtestdjd – find available allocatable memory, custom new handler. (Based on C code by DJ Delorie.)

runerr – elementary example of `std::runtime_error` exception, and throwing an exception.

sfx – custom exception class, multiple catch blocks.

tempfib – bizarre example from a C++ Standards Committee defect report. Uses recursive templates to compute Fibonacci sum. This is an example of what cutting-edge C++ programmers are up to.

testcomplex – fun and games with the `complex` type.

testcon – fun with constructors: taking care of failure to allocate inside a constructor, and some other things of interest. See comments for a suggested exercise.

testmin – custom support for template `min`; specialization of `std::ostream`.

testtbuf – exercises the `tbuffer` template discussed in the text.

teststrm – C++ version of `strtod` to compare against *your* solution!