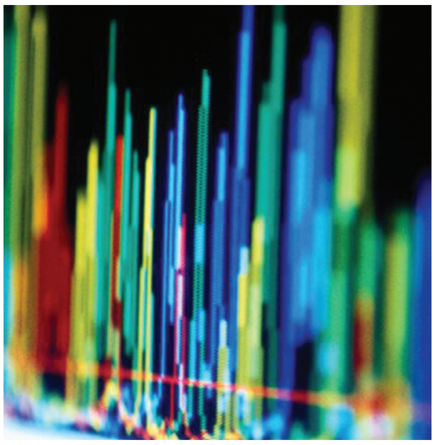
WaveTools

Canutations

In our February issue we took a detailed look at typical fields of application for WaveTools, particularly the wfct, wflt, wmix, and wplot programs. In this issue we will be looking at the remaining tools and demonstrating their capabilities.

BY VOLKER SCHMITT



We generated various waveforms (sine, triangular, rectangular, sawtooth, and noise) while working with the *wfct* program, however, there is one option that we have completely ignored so far: the parameter *-p* that allows you to define the phase shift for the wave you are creating. For a better understanding of phase shifting, let's think back to a simple formula from our technology lessons: as you know, *sin(x + 1/2PI) = cos(x)*. If:

```
wfct 440Hz 10ms
```

creates a sine wave, you can use:

```
wfct -p90deg 440Hz 10ms
```

to create a cosine wave (note that *1/2PI* corresponds to *90°* – , however, you can also use a radian *-p1.570796rad* to define *1/2PI = 1.570796*). Now, remembering those addition theorems:

```
1/2(sin a + sin b)=sin(1/2(a+b)⮐
)x cos(1/2(a-b))
```

you might be interested in verifying this graphically. To do so, replace *a* with *400Hz* and *b* with *480Hz*, and then create *wav* files for the two terms on the left and right of the equals sign. The first term is defined by:

```
(wfct 400Hz 30ms; wfct 480Hz ⮐
30ms) | wmix -s - -
```

Note that the *-s* flag is used to divide by the number of input files. We now require *1/2(a + b) = 440Hz* and *1/2(a-b) = 40Hz* for the second term, and can use:

```
(wfct 440Hz 30ms; wfct 20Hz ⮐
30ms) | wmix -m - -
```

to generate this. The parameter *-m* is used to perform multiplication. A quick look at Figures 1 and 2 shows that they really do look identical.

What does it do?

The *winf* tool displays header information for *wav* files. If you do not specify any options, only the length, the sampling frequency, and the resolution (in bits) of the file is output. This can be quite interesting.

Imagine that you have just downloaded your favorite tune from the Web – you could then use *winf* to find out more about quality of the recording. *wav* files in particular tend to be low quality – using slower sampling rates or a lower resolution to avoid using huge amounts of storage space. Let's use our cosine wave as an example, and type:

```
wfct -p90deg 400Hz 1s ⮐
-ocosinus.wav; winf cosinus.wav
```

to display the required information on screen:

```
cosinus.wav : time 0:01.00 s ( ⮐
11025 values), sampling ⮐
11025 Hz  8 bits
```

It's not really surprising to see that the sampling frequency is set to 11025 Hz by default, that is, if you do not use the option *-s* to change this. If you leave out the *-b* flag, the resolution defaults to 8 bits.

However, the *-s* flag for the *winf* program, which displays some statistics, may be more interesting. The output shows both minimum and maximum values and the volume for the file, expressed as a percentage of the sine

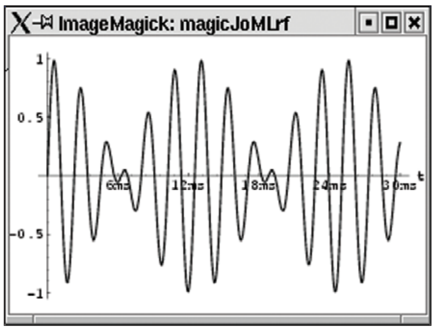


Figure 1: (wfct 400Hz 30ms; wfct 480Hz 30ms) | wmix -s - - | wplot | display

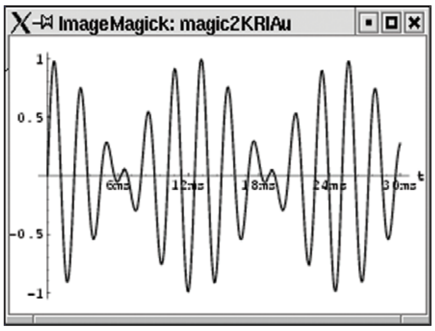


Figure 2: (wfct 440Hz 30ms; wfct -p90deg 40Hz 30ms) | wmix -m - - | wplot | display

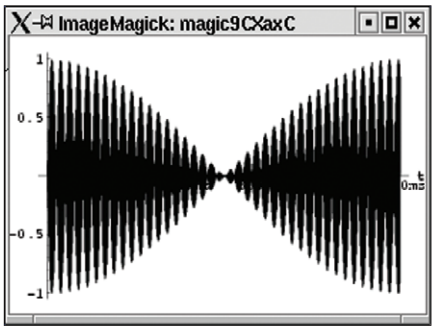


Figure 3: (wfct 440Hz 1s; wfct 439Hz 1s) | wmix - - | wplot | display

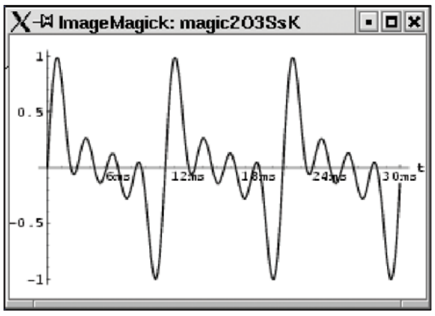


Figure 4: (wfct 100Hz 30ms;wfct 200Hz 30ms;wfct 300Hz 30ms;wfct 400Hz 30ms) | wmix - - - - | wplot | display

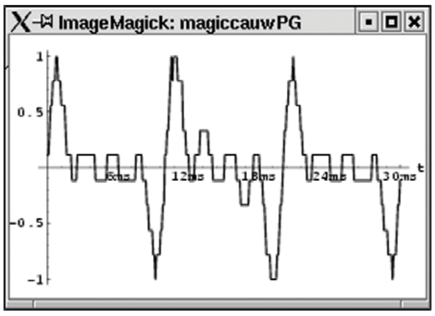


Figure 5: (wfct 250Hz 30ms;wfct 200Hz 30ms;wfct 50Hz 30ms | winv) | wmix -m - - - | wflt -n | wplot | display

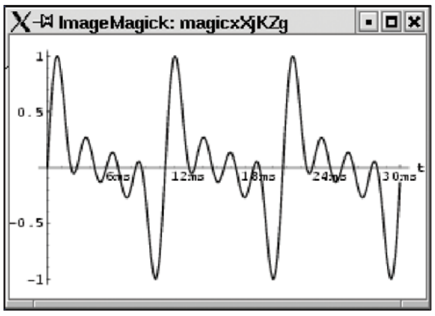


Figure 6: (wfct -b16 250Hz 30ms;wfct -b16 200Hz 30ms;wfct -b16 50Hz 30ms | winv) | wmix -m - - - | wflt -n | wplot | display

wave. Were you aware that overlaying a sine and a cosine wave will retain 99.4% of the volume of the sine wave, whereas overlaying *sin(x)* with *sin(x + 3/4PI)* drops down to 76.5% of the sine wave's original volume (cf. Listing 1)?

Didn't think so. But just take a look at the wave to find out why.

The result of overlaying waves with an almost identical frequency is also interesting. There is some interference (Figure 3 shows a 440 Hz and a 439 Hz wave overriding each other once a second), and thus the sum of these two waves will not produce the volume of the sine wave.

```
(wfct 440Hz 1s; wfct ↩
439Hz 1s) | wmix - - | winf
```

returns 70.5% of the sine wave volume.

The Microscope

The *wview* program can display *wav* files interactively on-screen. You can use the up and down arrows to change the interval for the data to be displayed. The left and right arrow keys allow you to scroll the selection window along the time axis. Additionally, you can use [PgUp] and [PgDn] to zoom the display. As the program runs in SVGA mode, you need to press [Q] or [Ctrl + c] to quit.

Division

As an extra bonus, this issue's subscription CD includes the *winv* program. This program is not part of the *WaveTools* collection; I wrote it myself to add division facilities to the addition and multiplication facilities provided by the *wmix* program. The range of amplitude values is between -1 and 1. Inverting these values would place them out of range. Thus, *winv* normalizes its output by restricting peak values to 1.

The subscription CD includes the statically linked *winv* binary. You can copy this file to */usr/local/bin* (*cp .../winv /usr/local/bin*). The program supports all the important output facilities provided by the other *WaveTools*. To test *winv* reapply the mathematical formula used for calculations with complex numbers; the formula is shown below.

$$\sin\varphi + \sin 2\varphi + \sin 3\varphi + \ldots + \sin n\varphi = \frac{\sin\frac{n+1}{2}\varphi \sin\frac{n}{2}\varphi}{\sin\frac{\varphi}{2}}$$

We set n = 4 and chose an angle of 100 Hz. We then created the left side of the equation as follows:

```
(wfct 100Hz 30ms; wfct ↩
200Hz 30ms; wfct 300Hz 30ms; ↩
wfct 400Hz 30ms) | wmix - - - -
```

and used *winv* as the expression in the denominator of the equation, which was then created as follows:

```
(wfct 250Hz 30ms; wfct ↩
200Hz 30ms; wfct 50Hz 30ms | ↩
winv) | wmix -m  - - - | wflt -n
```

Finally, we used *wflt -n* to normalize the results – this reverted the normalization performed by *winv*. If we now look at the expression on the left in Figure 4 and compare it with the expression on the right in Figure 5, we can say that the equation more or less worked.

However, the somewhat angular appearance of these figures means that there is some numeric instability. To resolve this issue we can up the sampling rate and resolution to 16-bit using the *-b16* option. The result shown in Figure 6 is far more satisfactory and can no longer be distinguished from Figure 4. ■

Listing 1: *winf* Output

```
volker@mouse: (wfct 440Hz 1s; wfct -p90deg 440Hz 1s) | wmix - - |
  winf -s
-standard-input-    : time 0:01.00 s (  11025 values), sampling 11025 Hz
  8 bits
  Amplitude (x[n])  : min = -1.00000  mid = -0.00931  max = +0.98438
  Step (x[n+1]-x[n]): min = -0.25781  mid = +0.15814  max = +0.25781
  Zero Values       : n =     880  n/sec =   880.0
  Extreme Values    : n =     880  n/sec =   880.0
  Standard Deviation (Volume): s = 0.70295 = 99.4 %

volker@mouse: (wfct 440Hz 1s; wfct -p135deg 440Hz 1s) | wmix - - |
  winf -s
-standard-input-    : time 0:01.00 s (  11025 values), sampling 11025 Hz
  8 bits
  Amplitude (x[n])  : min = -0.77344  mid = -0.00779  max = +0.75781
  Step (x[n+1]-x[n]): min = -0.20312  mid = +0.12194  max = +0.19531
  Zero Values       : n =     880  n/sec =   880.0
  Extreme Values    : n =     880  n/sec =   880.0
  Standard Deviation (Volume): s = 0.54108 = 76.5 %
```

INFO

[1] WaveTools: *http://tph.tuwien.ac.at/~oemer/wavetools.html*