Using waveforms in Faust- Tutorial

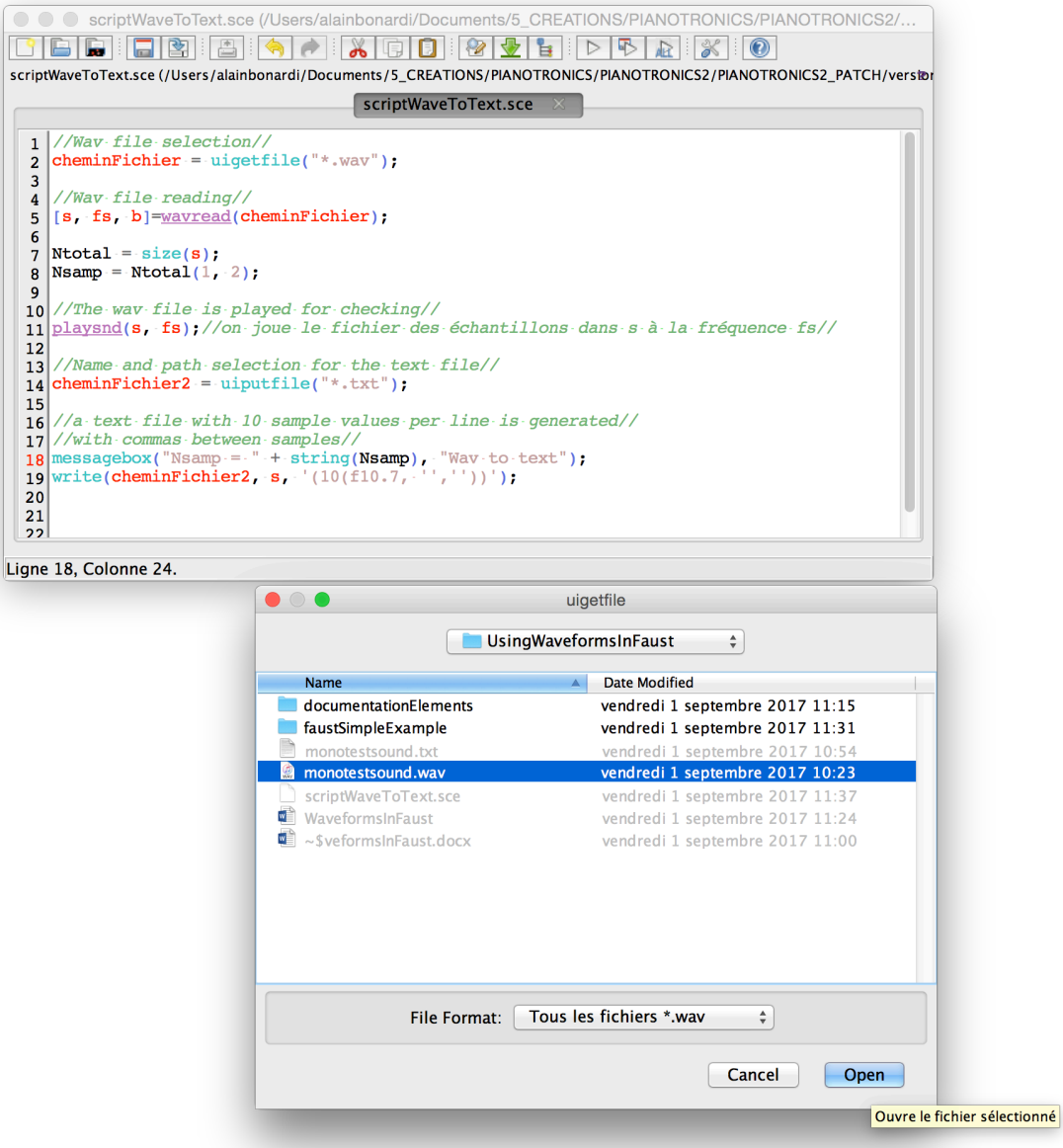
Alain Bonardi

Objectives

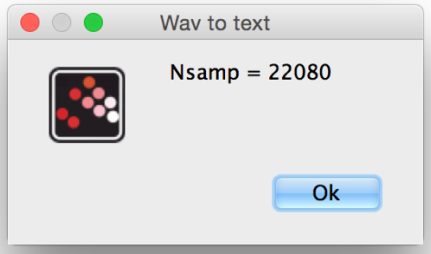
The purpose of this tutorial is to easily generate and use waveforms in Faust language. We start from a mono Wav file, and the idea is to fetch the sample values and be able to build a Faust code including a looper using this waveform. We will use Scilab, Max 7 and a text editor.

Converting a mono WAV file to a text file

Open the *scriptWaveToText.sce* script in Scilab (from SciNotes application).
Launch the script.



Select a mono WAV file, for instance *monotestsound.wav*
It is then played (for checking) by the Scilab script. Then select a name and location for the text file that will contain the samples, for instance *monotestsound.txt*
At the end, the Scilab script indicates the number of samples written, for instance in this case 22080:



The generated file contains the list of sample values separated by commas. Here are for instance the first two lines of the *monotestsound.txt* file:

```
0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000305,-0.0000610,
0.0000610,-0.0000305, 0.0000000, 0.0000000,
0.0000000, 0.0000305,-0.0000610, 0.0000610,-0.0000610, 0.0000610,
0.0000000,-0.0000916, 0.0001221,-0.0001221,
```

and the last two lines:

```
-0.0001221,-0.0002747,-0.0000610,-0.0002441,-0.0001221,-0.0001526,-
0.0001526,-0.0000916,-0.0001221,-0.0000610,
-0.0001221,-0.0000305,-0.0001221, 0.0000305,-0.0001221, 0.0000305,-
0.0000610,-0.0000305, 0.0000305,-0.0000610,
```

To make it usable in Faust, just duplicate this file and rename the copy for instance *monotestsoundfaust.txt*. Then, before the first sample just add two lines:

```
Nsamp = 22080;
mySamples = waveform {
```

The beginning of the file becomes:

```
Nsamp = 22080;
mySamples = waveform {0.0000000, 0.0000000, 0.0000000, 0.0000000,
0.0000305,-0.0000610, 0.0000610,-0.0000305, 0.0000000, 0.0000000,
0.0000000, 0.0000305,-0.0000610, 0.0000610,-0.0000610, 0.0000610,
0.0000000,-0.0000916, 0.0001221,-0.0001221,
```

At the end of the file, just replace the last comma by };

The last two lines become:

```
-0.0001221,-0.0002747,-0.0000610,-0.0002441,-0.0001221,-0.0001526,-
0.0001526,-0.0000916,-0.0001221,-0.0000610,
-0.0001221,-0.0000305,-0.0001221, 0.0000305,-0.0001221, 0.0000305,-
0.0000610,-0.0000305, 0.0000305,-0.0000610};
```

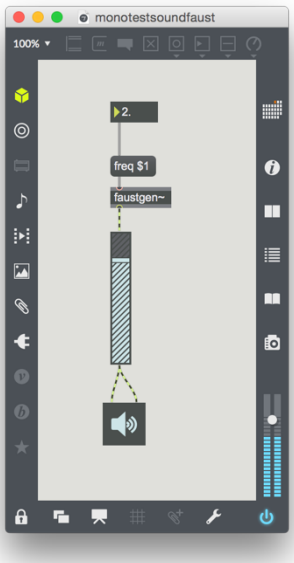
Creating a faust process using the text file of samples

Raw player

We can then copy this file into a Faust code file enabling to play it as a loop at any frequency. Here we created a new file called *monotestsoundfaust.dsp*:

```
import("stdfaust.lib");
freq = vslider("freq", 2, 0.01, 48000, 0.01);
Nsamp = 22080;
mySamples = waveform {0.0000000, 0.0000000, 0.0000000, 0.0000000,
0.0000305,-0.0000610, 0.0000610,-0.0000305, 0.0000000, 0.0000000,
0.0000000, 0.0000305,-0.0000610, 0.0000610,-0.0000610, 0.0000610,
0.0000000,-0.0000916, 0.0001221,-0.0001221,
…

//all the sample values

…
-0.0001221,-0.0002747,-0.0000610,-0.0002441,-0.0001221,-0.0001526,-
0.0001526,-0.0000916,-0.0001221,-0.0000610,
-0.0001221,-0.0000305,-0.0001221, 0.0000305,-0.0001221, 0.0000305,-
0.0000610,-0.0000305, 0.0000305,-0.0000610};

pdPhasor(f) = os.phasor(1, f);

elementaryPlayer(f0, mySamp, mySampNum) = myPlayer
      with {
                  zeroToOnePhase = pdPhasor(f0) : ma.decimal;
                  myIndex = zeroToOnePhase * float(mySampNum);
                  i1 = int(myIndex);
                  s1 = (mySamp, i1) : (+(1), _, _) : rdtable;
                  myPlayer = s1;
};

process = elementaryPlayer(freq, mySamples, Nsamp);
```

We implemented this code in a Max 7 patch (*monotestsoundfaust.maxpat*), using faustgen~ object, with a control of the looping frequency.



The *elementaryPlayer* function in the Faust code is a raw looper without any interpolation, it can generate artifacts. But it can be directly used for instance for envelopes.

Linear interpolated player

We developed a linear interpolated player that enables to loop sounds with a good quality. The Faust code is in *monotestsoundfaust2.dsp*:

```
import("stdfaust.lib");
freq = vslider("freq", 2, 0.01, 48000, 0.01);
Nsamp = 22080;
mySamples = waveform {0.0000000, 0.0000000, 0.0000000, 0.0000000,
0.0000305,-0.0000610, 0.0000610,-0.0000305, 0.0000000, 0.0000000,
0.0000000, 0.0000305,-0.0000610, 0.0000610,-0.0000610, 0.0000610,
0.0000000,-0.0000916, 0.0001221,-0.0001221,
…

//all the sample values

…
-0.0001221,-0.0002747,-0.0000610,-0.0002441,-0.0001221,-0.0001526,-
0.0001526,-0.0000916,-0.0001221,-0.0000610,
-0.0001221,-0.0000305,-0.0001221, 0.0000305,-0.0001221, 0.0000305,-
0.0000610,-0.0000305, 0.0000305,-0.0000610};

pdPhasor(f) = os.phasor(1, f);

linearInterpolatedPlayer(f0, mySamp, mySampNum) = myPlayer
      with {
                  zeroToOnePhase = pdPhasor(f0) : ma.decimal;
                  myIndex = zeroToOnePhase * float(mySampNum);
                  i1 = int(myIndex);
                  i2 = (i1+1) % int(mySampNum);
                  d = ma.decimal(myIndex);
                  s1 = (mySamp, i1) : (+(1), _, _) : rdtable;
                  s2 = (mySamp, i2) : (+(1), _, _) : rdtable;
                  myPlayer = s1 + d * (s2 - s1);
};

process = elementaryPlayer(freq, mySamples, Nsamp);
```

Looping frequency

For both players the frequency to play at the origin speed of the Wav file is defined by the ratio *SR / Nsamp* where *SR* is the sampling rate and *Nsamp* the number of samples of the waveform (in our example, 22080). Rather than adjusting this frequency, we could handle a multiplier *freqmult*, having a final frequency defined by *freqmult * SR / NSamp*. This means that we get the original speed for *freqmult = 1*, half the speed for *freqmult = 0.5*, twice the speed for *freqmult = 2*, etc. This is implemented in *monotestsoundfaust3.dsp*:

```
import("stdfaust.lib");
freqmult = vslider("freqmult", 1, 0.01, 100, 0.01);
…
origFreq = ma.SR / Nsamp;
…
```

```
process = linearInterpolatedPlayer((freqmult*origFreq), mySamples, Nsamp);
```

The patch is therefore adapted (monotestsoundfaust3.maxpat):