

Using FOF-derived opcodes in granular processing

The function table used by FOF to create its grains is usually a sinewave when this opcode is used on formant synthesis. However, if we instead we use a wave with more harmonics, we will create formants at multiples of the selected formant frequency. For instance, if we use a wave with 3 harmonics (1, 2 and 3 times the fundamental), and formant frequency 500 Hz, the three formant regions will be at 500, 1000 and 1500 Hz (1,2 and 3 times that frequency). Of course, this means also that if we use a wave with several harmonics, we might run into aliasing problems. Consider using a wave with 10 harmonics and formant frequency of 3000 Hz. The highest formant region will then be 30 KHz, well above the usual Nyquist frequency of 22.5 KHz. Using other waveforms as inputs, such as sampled-sound GEN01-generated function tables, can cause similar situations. The aliasing result will of course, depend on the *xform* parameter of the FOF opcode, the formant frequency. If that frequency is $SR/table_length$, then the sample will be played at the original rate and no aliasing will occur (provided of course that there is no aliasing in the original sample!).

The best use of FOF opcodes in conjunction with sampled-sound tables is to process sound with granular techniques. The original FOF opcode is somewhat limited and do not provide the full range of parameters needed for granular synthesis, so two derived opcodes were developed to process input samples using this technique. These two opcodes are FOF2 and FOG.

FOF2 is very similar to FOF, except that the i-time FOF parameter *iphase* is substituted by the k-rate *kphase*, making it possible to start the table readout at different positions during performance. FOF2 also has an extra k-rate parameter (instead of FOF's *imode*), *kgliss*, which provides a k-rate glissando control for each grain, in octaves relative to the start pitch. For instance, a value of 2 will create an upward octave glissando and -2 a downward one. Other interval ratios can also be used.

FOG is also similar to FOF and FOF2. The difference is basically that a new a-rate parameter for the start phase (or position) is introduced (instead of *kphase*) and a grain pitch transposition factor is in the place of the usual formant frequency parameter *kform*. The pitch transposition value is a simple multiplier (1, no change, 2 octave upwards, 0.5, octave downwards, etc.). The relationship between the old formant frequency and the pitch transposition factor is

$$formant_freq = pitch_transposition \times \frac{sampling_rate}{table_length}$$

where the sampling rate is that of the sampled-sound source and the table length refers to the function table *ifna*.

FOF2 and FOG are normally used in synchronous granular resynthesis, in conjunction with sampled sounds stored in a function table using GEN01.

Timescale and pitch modification using FOG

One of the most basic uses of granular resynthesis is to modify pitch and duration (timescale) independently. By chopping up an input sound in little grains and recombining them we can stretch or compress its playback. By altering the pitch of each individual grain, we can, independent of time, transpose it upwards or downwards.

The following example uses the FOG opcode in a timestrech/compress and pitch transposing instrument:

```
; Independent pitch / timescale modification
; using granular resynthesis
; VL, 2001

sr=44100
kr=4410
ksmps=10
nchnls=1

instr 1

itime = 1.8           ; time ratio
ipitch = 1.5         ; pitch ratio

isampdur = ftlen(1)/sr ; sample duration
ifindur = itime*isampdur ; final duration
iolaps = 2           ; grain overlaps
iamp = 10000        ; amplitude
igdur = 0.02        ; grain duration
idens = (1/igdur)*iolaps ; grain density (grains/sec)

aspd phasor 1/ifindur ; read pointer pos (0 - 1)

; FOG opcode, similar to FOF, except for the parameter 'aspd'
; 'kform' is also substituted by 'kpitch', a pitch transposition ratio
; the relationship between FOF's kform and FOG's kpitch is
; kform == kpitch * (ftsr(ifna)/ftlen(ifna))

asig fog iamp, idens, ipitch, aspd, 0, 0, 0.003, igdur, 0.005, iolaps, 1, 2, p3

    out asig

endin
```

The first two parameters set the timescale and transposition ratios, independently of each other:

```
itime = 1.8           ; time ratio
ipitch = 1.5         ; pitch ratio
```

In order to calculate the final duration of the sound, we have to find out how long it is (in seconds) and then multiply it by the timescale ratio:

```
isampdur = ftlen(1)/sr ; sample duration
ifindur = itime*isampdur ; final duration
```

The grain duration has to be small, around 20 milliseconds, so that we do not hear the echo-like grain repetitions. We can also set-up the number of overlapping grains, 2 for the minimum of 1 overlap (of two grains). These parameters then are used to set the density of grains per second, which is the inverse of the duration of the grain times the number of overlaps:

```
iolaps = 2           ; grain overlaps
igdur = 0.02        ; grain duration
idens = (1/igdur)*iolaps ; grain density (grains/sec)
```

If there is no overlap, then the density is just the inverse of the duration of the grains, so that one grain is followed by another in the stream, in this case 50 grains/sec. In the case above, there is an overlap of 2 grains, so that the density is twice that number, 100 grains/sec.

The next step is to make the read pointer position, which varies normalised between 0 and 1, to increase in the time interval of the final duration. In other words, the read pointer will progress from 0 to 1, from reading at the beginning of the table to reading at the end of the table, in *ifindur* seconds. This can be a one-shot situation or we can loop it so that when it reaches the end of the table it goes back to the beginning. In order to do so, we use the **phasor** opcode that does just that: varies from 0 to 1 at a certain rate (per second); and that rate is, of course, the inverse of the final duration, $1/\text{ifindur}$.

```
aspd phasor 1/ifindur ; read pointer pos (0 - 1)
```

Finally, we use the FOG generator to process the sample-sound source:

```
asig fog iamp, idens, ipitch, aspd, 0, 0, 0.003, igdur, 0.005, iolaps, 1, 2, p3
```

Note that the *xfund* parameter in FOF is the *idens* parameter here, just the number of grains generated each second. Also in relation to the local envelope, the values used are: for *kband*, which controls the exponential decay rate for each grain, we use 0 meaning 'no decay'; for *kris*, the rise time of each grain, we use 3 milliseconds; and for *kdec*, the tail-off decay time (here effectively the decay time of each grain), we use 5 milliseconds. *Kband* is set to 0 because we do not want the grains to fade away exponentially in the course of their duration. Remember: the higher *kband*, the faster the decay rate.