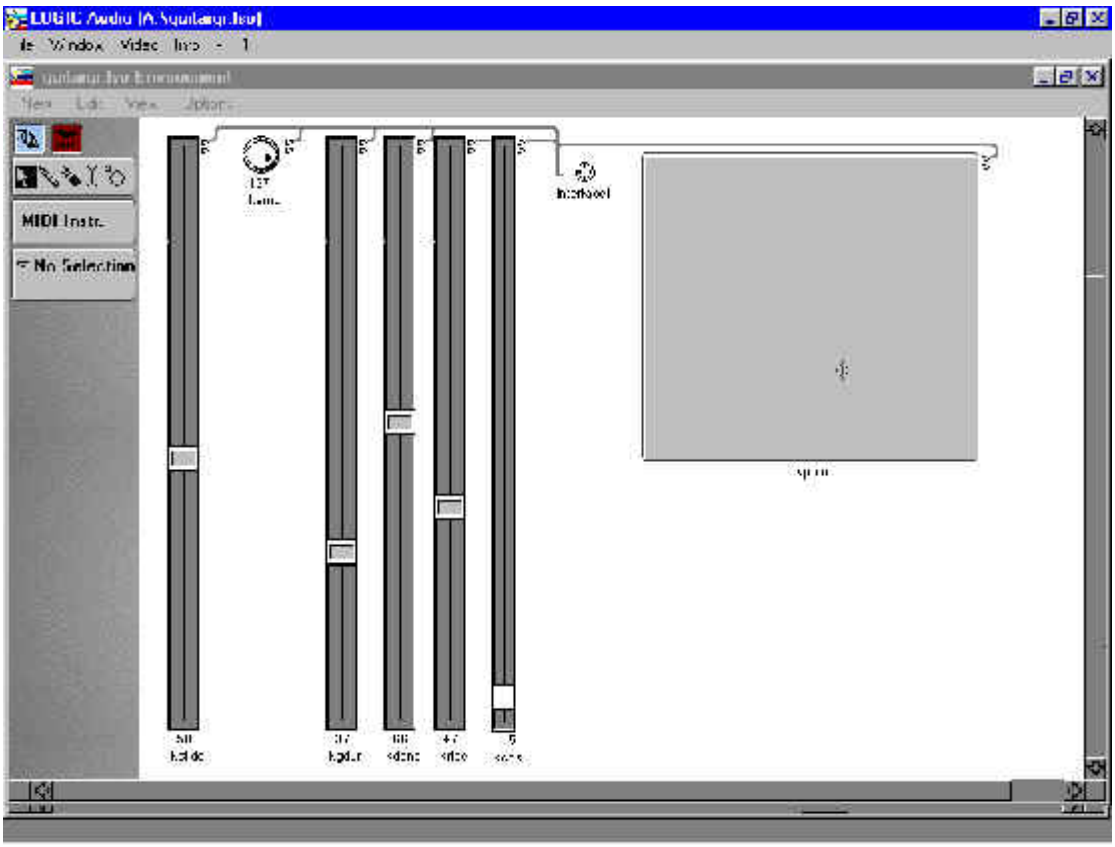
Controlling FOG parameters in realtime using DirectCsound

by Richard Bowers

fog-slider.csd fog-slider2.csd

The Csound **fog** opcode allows the user to apply granulation techniques to audio data held in a table. The soundfile can be read through, or arbitrarily accessed, at a variable rate while the pointer scans ahead to pick up enough data to produce each 'grain' of sound. This means that two speed parameters are being applied at once: firstly, the overall speed of reading through the file and, secondly, the speed of the pointer from the current location. It is possible to think of this in terms of sweeping a floor – a general movement across the floor coupled with short sweeping motions as you go.

It is quite possible to control the **fog** parameters in realtime via a MIDI controller in hardware or software. For the second movement of my *'Incident'* for guitar and tape I wanted to generate some textures using material from the first movement. [extract]. I used Emagic's Logic Audio, which provides a range of graphical sliders, as the interface to the Csound orchestra. (See screenshot below for a view of Logic's sliders). This was run on a i486 and the data carried over standard MIDI cables to my faster Pentium PC running Direct Csound. You may be able to achieve this on a single machine using Hubi's MIDI Loopback Device [http://www.geocities.com/SiliconValley/Vista/2872/hmidilb/hmdlpbk.html] as described by Hans Mikelson in Csound Magazine (Summer 1999).



Alternatively - and, perhaps, preferably - it is possible to control the parameters using the GUI controllers (DirectCsound 5.10). This method is described later.

Furthermore, I chose to process 'live' audio signals. This is described later.

Using MIDI data

MIDI is a well-defined system capable of controlling Csound instruments thanks to Csound's range of supporting opcodes. MIDI's limitations are something you learn to live with. However, the fact that practically everything operates within the range 0-127 is at least clear (if not very powerful) and you are forced to focus attention on the way Csound handles such limited data.

The low resolution of MIDI data – 128 possible values - is insufficient for the smooth operation of the **fog** opcode. This was remedied within the orchestra by using the **slider8f** opcode to provide interpolation between adjacent MIDI values. (However, despite interpolation the values will always rest on an integer between 0 and 127 – the interpolation merely smooths the *transitions* between integers).

All of the MIDI controllers operate in the range 0-127 so value conversion was necessary using the *minimum* and *maximum* parameters in **slider8f** to create useful ranges for **fog**. These conversions were different depending on the context.

```
gktrans, gkbw, gkpoint1, gkpoint2, gkgdur, gkdens, gkenv, gkcps slider8f 1, \
1, 0.25, 4, 1, 0, ico,\
2, 0, 16, 10, 0, ico,\
3, 0, 1, 0, 0, ico,\
4, 0, 1, 0, 0, ico,\
5, .01, .1, .01, 0, ico,\
6, 1, 254, 127, 0, ico,\
7, 0, .5, .25, 0, ico,\
8, .001, 3, 1, 0, ico
```

We now have the range 0-127 converted to the ranges which are useful to the various parameters in **fog**.

fog

The manual gives the following entry for the **fog** opcode

ar **fog** xamp, xdens, xtrans, xspd, koct, kband, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur[, iphs[, itmode]]

I used the various MIDI controllers for adjusting the **fog** parameters as follows:-

```
ctrl1=gktrans =xtrans
ctrl2=gkbw =kband
ctrl3=gkpoint1 =xspd (after modification)
ctrl4=gkpoint2 =ditto for second fog (instr 3)
ctrl5=gkgdur =kdur
ctrl6=gkdens =xdens
ctrl8=gkenv =kdur & kdec (in combination with gkgdur)
ctrl9=gkcps =xspd
```

A problem with operating **fog** in realtime is the computational load created by increasing the values of certain parameters in the opcode. In a realtime situation the result of overloading the system is choppy sound, which is normally to be avoided. With the **fof** family of opcodes it is possible to crash the program in certain cases of overload. In my setup it was apparent that the increase in grain duration beyond a certain point had little effect on the timbre but increased the load because of the excessive overlapping of grains. By keeping the upper limit to 0.1 seconds it allowed for a larger range of values for the *density* parameter which has a more marked effect on the timbre

A method of constraining the load was achieved by using the sliders' output values in two ways simultaneously for two parameters of the opcode: a value of, say, 30 can produce two complementary values, 30 and 97 if we deduct the value from the overall range. (Remembering to take the converted range, not the 0-127 range).Thus, a single slider can increase one parameter whilst reducing another – keeping the load roughly constant. An example of this method in my orchestra is where the *density* parameters for the two separate **fog** opcodes are controlled by the **slider8f** output **gkdens** and **254-gkdens** respectively.

With these limits in mind – and they will vary depending on the speed of your system – you can experiment with the settings so that, ultimately, the sliders can be freely manipulated without introducing unwanted glitches, discontinuities or crashes.

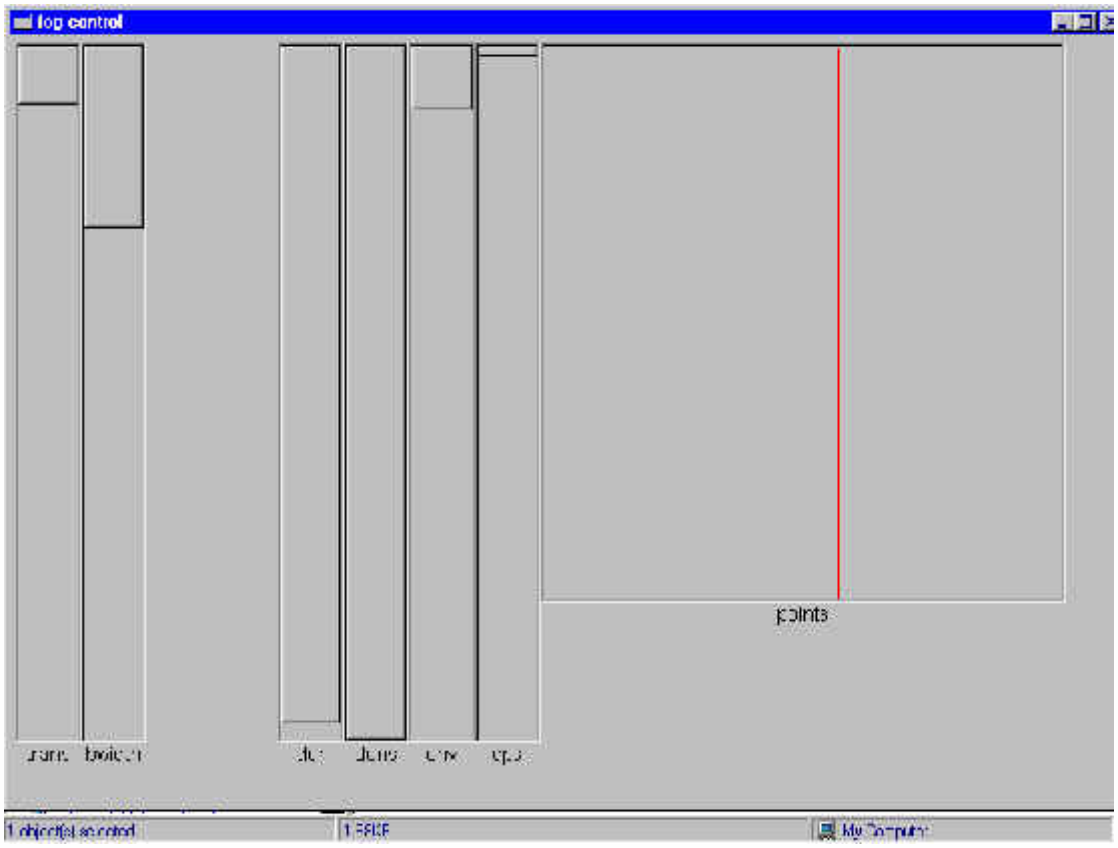
The MIDI controllers

One type of graphical slider provided within Logic Audio is a 'vector' slider. This is essentially a square with a range of values running along the x axis and another range along the y axis. It enables the user to control, with the mouse, two independent MIDI controllers (or the same controller on different MIDI channels). You may, for instance, use the x-axis to control the **xtrans** parameter and the y axis for the **xspd** parameter. In my instrument I used it to control the location of the pointer for the two independent **fog** opcodes. This meant I could 'scrub' through the two soundfiles simultaneously using the vector slider.

It is feasible to extend the use of vector sliders to pair-up different combinations of parameters. For example, I may pair-up the grain density with the grain duration on one slider while on another grain density is paired with grain speed. This expands the range of gestures available although it may compromise the constraint technique mentioned above. Experiment is the key to finding the right combinations.

An alternative: using the DirectCsound GUI controllers

It is now possible to apply similar graphical controllers using a new set of opcodes implemented by Gabriel Maldonado in his DirectCsound (v. 5.10).



This can all be carried out on a single pc. Changing the code is simple. Instead of using the **slider8f** opcode, we now have (in the orchestra header) the following lines:-

```
FLpanel "fog control", 450, 550, 10, 10; <---opens the panel to place the sliders
gktrans, i1 FLslider "trans", .25, 4, 0, 2, -1, 50, 500, 5, 5
gkbw, i2 FLslider "bwidth", 0, 16, 0, 2, -1, 50, 500, 55, 5
gkpoint1, gkpoint2, i3, i4 FLjoy "points", 0, 1, 0, 1, 0, 0, -1, -1, 400, 400, 40
gkgdur, i5 FLslider "dur", .01, .1, 0, 2, -1, 50, 500, 105, 5
gkdens, i6 FLslider "dens", 1, 254, 0, 2, -1, 50, 500, 155, 5
gkenv, i7 FLslider "env", 0, .5, 0, 2, -1, 50, 500, 205, 5
gkcps, i8 FLslider "cps", .001, 3, 0, 2, -1, 50, 500, 255, 5
FLpanel_end
FLrun; <----required to run the GUI
```

This offers similar functionality, with greater ease and economy, to the MIDI sequencer option including the **FLjoy** opcode to replace Logic's vector controller.

To avoid a program crash on startup, I included the following initialization values in the header:

```
gktrans init .25
gkbw init 0
gkpoint1 init 0
gkpoint2 init 0
gkgdur init .01
gkdens init 1
gkenv init 0
gkcps init .001
```

Realtime and the p3 problem.

One thing we take for granted in the Csound notelist is the p3 (note duration) value. We use this value in our instruments for control purposes – to determine the durations in an amplitude envelope

or to control the speed of reading data in, say, a **pvoc** analysis file. This p-field can still have musical meaning in the realtime context but there are many instances when you cannot pre-programme the note data (eg. in instruments driven by MIDI notes it has no meaning because a p3 value is not transmitted) and some in which the p3 value is not in itself useful to the instrument in any 'musical' sense.

An example of the latter is where I have set the instruments to run continually by providing a large value for p3 in the score. This meant that I had to rethink the way I would normally read through the soundfile in **fog**. Ordinarily (ie when I would have used p3 as the 'musical' note duration), I may have set a **line** to direct the pointer through the soundfile as follows:

```
kpoint line 0, p3, 1
```

resulting in a smooth read through the file over the note's duration. In my realtime version I wanted to continually and repeatedly read through the soundfile while allowing for control of the rate of reading. So, I used the **phasor** opcode, which is ideally suited because it generates a range of values from 0 to 1 which matches the range used for reading through the soundfile from beginning to end. Furthermore, the speed at which it does this can be controlled using **gkcps** as follows:

```
ap phasor gkcps
apoint=(gkpoint1)+ap
```

and

```
ap phasor gkcps
apoint=(gkpoint2)+ap
```

where **gkcps** is the speed of the overall read through the table, **ap** is the pointer position produced by the phasor and **gkpoint1&2** is an offset value from which the pointer reads. The resulting **apoint** is the real pointer-position in the table. This pointer position is used by **fog** in the xspd parameter.

Processing realtime audio input

One goal I set myself was to pour realtime audio into a dynamically changing table. This meant that the table was constantly changing its content – producing new sonic material for granulation.

First the audio is fed in in the usual way using **ins**. Then a previously allocated table (I used Gen10 but there may be a better one to use) is indexed using a phasor at a rate equavalent to the audio rate:

```
and phasor 1/(ilength/sr)
andex=and*ilength
```

This indexing is applied to a table-writing (**tablew**) opcode in wrap-mode. This means that the indexing will wrap back to the start of the table when it reaches the end. In other words, the **tablew** simply cycles through the table - writing audio data as it arrives – at the audio rate.

```
tablew asig, andex, ifn, 0, 0, 1
```

With this process in place the fog opcodes (and any others that access tables) are free to pick up the audio on-the-fly. (Bear in mind that there is a 'moving join' where old data is being overwritten with new).

That covers the bare bones of the **fog** opcode and how it can be controlled 'live' via graphical sliders and I hope it at least gives a hint of the larger scope for developing quite sophisticated gestural control. It may also be used as a way of testing the parameters to gain desired values for presets (e.g.

the switches in the GUI opcodes).

Richard Bowers – Cardiff, Wales - April 2001

richard.bowers@ntlworld.com

http://www.kakutopia.fsnet.co.uk/