

A comparative review of three audio processing environments

Jamie Bullock
Birmingham Conservatoire, UK

August 4, 2006

Abstract

Three simple tasks were contrived, in order to make a basic comparison between Csound (5.0), PD (0.39), and SuperCollider server (aka. SuperCollider 3). The tasks involved polyphonic synthesis, processing of live input, and real-time control using OSC and graphical input widgets. The purpose of the exercise was to gain an overview of how each environment handled the tasks, and make conclusions about each environment's strengths and weaknesses.

1 The tasks

The tasks were as follows:

1.1 Task 1

Create a simple band-limited sawtooth synthesiser that provides 3 note polyphony, and can be played by a MIDI keyboard in real time. The synth must provide an ADSR envelope with, which sustains until a note-off or zero velocity MIDI event triggers the release. There should be no perceptible aliasing. Envelope times should be as follow:

- Attack: 100ms
- Decay: 100ms
- Sustain level: 60
- Decay time: 1000ms

The envelope must be non-linear.

1.2 Task 2

Create a sine wave oscillator whose frequency can be controlled by a graphical slider. The 'slider value' must be sent to the oscillator using the OSC protocol, with a user defined address space. For example, the slider could send '/synth1/freq <frequency value>'

1.3 Task 3

Create a signal processor that takes a live input, e.g. from a microphone, and delays the signal by a variable amount (between zero and two seconds). There must also be variable feedback (between 0 to 100%) from the output of the delay back to its input. The delay time and feedback should be controllable in realtime, with interpolation to avoid zipper noise.

1.4 Common requirements

The patches created to accomplish the above tasks should run without modification on both Mac OS X, and Linux operating systems.

2 Assessment criteria

The following criteria were used to evaluate each piece of software:

- Usability: how satisfying was it to use the software? This includes the time taken to complete the task, the perceived usefulness of supporting materials (examples and documentation), and the overall 'feel' of the software.
- Efficiency: how efficient was the software in achieving each task? This includes the amount of user input (mouse clicks, key presses) required to complete the task. It also includes the amount of processing power (CPU utilisation) required to run each task.
- Appropriateness: how appropriate was the piece of software for each task? This includes availability of internal opcodes/ugens and requirements for third party software.
- Uniformity: how uniform was the experience of the software between platforms.

Table 1: Lines, characters and objects in patches

Task	Application	Lines	Chars	Objects
1	SC3	29	758	12
	Csound	12	158	5
	PD			158
2	SC3	22	546	8
	Csound	19	371	9
	PD			8
3	SC3	30	854	11
	Csound	26	473	12
	PD			11

3 Results

3.1 Efficiency

The table below summarises the lines of code, characters, and objects¹ used by each piece of software for each task.

It can be seen from the above table that, the three environments are roughly equivalent in terms of the objects used for each task, apart from PD, which used far more objects for the synthesis task. This is due to the fact that PD lacks built-in band-limited oscillators, envelope generators, and automatic polyphonic voice generation, so these things need to be archived by creating abstractions and manually instantiating voices.

Csound required far less user input than either of the other two environments, and in the case of the synthesis task, it required one third as many typed characters as SuperCollider. This is due to the fact that Csound is syntactically very compact. In some cases it is also necessary to be more explicit in SuperCollider than in Csound. In SuperCollider variables need declaring, line endings must be terminated with semi-colons, functions must be delimited by curly braces, and code blocks must be delimited by parentheses. PD (without externals) provides a 'low level' set of functionality, so more work is needed initially to create 'abstractions' to perform basic tasks. Csound does more 'behind the scenes' allowing the user to focus on higher level patch elements.

Table 2 shows the load times for various sections of the task set. The computer running Mac OS X was a Quad 2.0GHz G5, and the computer running Linux was a 2.0GHz Pentium M. The measurements were taken us-

¹An object in this case is counted as an instance of a class or Ugen

Table 2: CPU load

Task	Application	Platform	Idle load (%)	Active load(%)
1	PD	OS X	15-19	15-19
	Csound	OS X	0.14	0.2
	SC3	OS X	0.2	2-3
	PD	Linux	3-4	3-4
2	PD	Linux		
	Csound	Linux	N/A	0.3-0.7
	SC3	Linux	0.2	2-3
SwingOSC slider	SC3	Linux	N/A	1.7-2.3

Table 3: Time taken to create patches

Task	Application	Time taken (m)
3	PD	6
	SC3	25
	Csound	15

ing the Unix command: 'top'. Csound created the lowest CPU load, with PD creating the highest. This is partly due to the network messages generated for communication between the PD GUI and the Pd service. It is also worth noting that there is no difference in PD between an inactive or active synth. This is because voices are allocated when the synth is created, and are therefore 'on' constantly. This can be avoided by using [switch] to turn off unneeded voices, but since this wasn't required to complete the task, it wasn't added.

Also significant, is the load created by the SwingOSC slider. SwingOSC was used as the GUI components of the SuperCollider tasks since SuperCollider native GUI components are Mac OS only. SwingOSC, runs as a separate process under a Java virtual machine. The additional load could be due inefficient GUI rendering, or increased network traffic (OSC messages) output by the slider when moving.

3.2 Usability

The above table shows the approximate times taken to complete task 3 in each environment. The differences in the times can be accounted for by my relative familiarity with each piece of software, and the relative complexity of the patches.

Across all 3 tasks, I found Csound to be by far the most enjoyable to

use. The documentation is clear, concise and thorough and most of the DSP techniques involved in completing the tasks were 'hidden' by the opcodes. With each task, I felt as though I was engaging more creatively with the software when using Csound, because with each line of code musically, or visually meaningful parameters were made available.

The documentation for PD is good, although unless one is very familiar with the examples, it is often difficult to find relevant information quickly. Although PD required the most user input, it was relatively quick to work with, and particularly in the delay example, the visual nature of the software was helpful in showing 'signal flow'. PD's main drawback seemed to be the way in which parameters are coupled to objects. In complex patches (e.g. the sawtooth task), this can lead to important parameters being hidden in subpatches/abstractions, which then requires the user to take additional measures to make these parameters available for change.

I found the documentation in SuperCollider to be quite difficult to navigate through. The 'in-line' access to object help and class definitions is useful, but overall the documentation was not concise or structured enough to find the right information quickly. SuperCollider was the least enjoyable application to use for these particular tasks. In particular it seemed to require an unnecessary level of explicitness, particularly with regard to buffer management. A comparison between the Csound '.orc' files, and the SuperCollider '.sc' files in the appendices are telling in this respect.

3.3 Uniformity

The experience of PD on the Mac OS and Linux platforms was virtually identical. The way in which user preferences are saved is slightly different, but other than that, the look, feel and behaviour of the software was the same. This was also true of Csound, although, I was unable to get any of the Linux graphical editors to work, which although irrelevant to these tasks, was a slight annoyance. However, the choice of FLTK for Csound's own GUI widget facilities seemed excellent, providing a fast, 'lightweight' cross-platform GUI functionality with minimum fuss.

SuperCollider provided the least uniform user experience, with the GUI classes and the Document classes being Mac OS only. The SwingOSC bindings were therefore used to accomplish GUI based tasks, but this required the installation of additional software (SwingOSC and its dependencies). To use the SuperCollider '.rtf' files in my favourite editor (vim), it was necessary to run a script to convert them to plain text¹.

¹It is worth noting, that with the files in this format it is still possible to get syntax

3.4 Appropriateness

Csound was originally written to be used for non-realtime processing, so I was quite surprised how efficiently it handled these realtime tasks. However, although it is possible to do a lot in realtime with Csound, it doesn't run as an interpreter, so it lacks the 'live coding' element of Pure Data and SuperCollider. This wasn't a problem for these particular tasks, and I am aware that it is possible to achieve a level of scripting with TclCsound.

What did become a problem, was the lack of 'built-in' functionality in Pure Data. The sawtooth task would have been far easier using external objects that provide basic tools like band-limited oscillators and envelope generators. The most critical omission from PD's internals however, was the lack of OSC functionality. In order to complete task 2, it was necessary to use an external library.

As previously noted, it was also necessary to use an external application with SuperCollider to gain cross-platform GUI functionality.

4 Conclusions

The only piece of software that accomplished all three tasks on both Mac OS and Linux, with no change to the code, and no additional applications was Csound. Csound also required the least user input and provided the most comprehensive documentation. However, a lot of functionality was used in creating the SuperCollider and PD patches, that isn't available in Csound, and although it wasn't required in these tasks, it would be interesting to conduct further experiments that do require this functionality.

The sawtooth test was also conducted on Max/MSP. This was very similar to Pure Data in many respects, however, the task was accomplished a lot quicker due to the availability of built-in objects ([saw], [poly] etc.).

5 Further work

It would be useful to conduct these tests on Windows, as well taking additional timings. It would also be useful to conduct further tests, these might include:

- Phase Vocoder test
- Live coding test

highlighting and in-line help, as this functionality is handled by the editor

- Algorithmic synthesis
- Algorithmic event generation

It might also prove useful to make comparisons with other environments.

Appendix 1 - Sawtooth synth

Csound (sawsynth.orc)

```
nchnls = 18
```

```
instr 1
icps cpsmidi
iamp ampmidi 0dbfs*0.4

k1 mxadsr 0.1, 0.1, 0.6, 1.5
a1 vco2 k1 * iamp,icps
outch 15,a1,16,a1

endin
```

Csound (sawsynth.sco)

```
i1 0 600
```

Supercollider (sawsynth.sc)

```
MIDIClient.init(9,9);
MIDIIn.connect(8, MIDIClient.sources.at(8));

(
  var keys;

  SynthDef(\sawsynth,
    {
      arg freq=440, amp=0.2, gate=1;
      Out.ar([14, 15], Saw.ar(freq, EnvGen.kr(Env.adsr(0.1, 0.1, 0.6, 1,
                                                amp, -4), gate, doneAction:2)));
    }
  ).send(s);

  keys = Array.newClear(128);

  MIDIIn.noteOn = {arg src, chan, num, vel;
    var node;
    node = Synth.tail(s, \sawsynth, [\freq, num.midicps, \amp, vel/255]);
    keys.put(num, node);
  };

  MIDIIn.noteOff = {arg src, chan, num, vel;
    var node;
    node = keys[num];
    keys.put(num, nil);
    s.sendMsg("/n_set", node.nodeID, "gate", 0);
  };
)
```


Appendix 2 - OSC synth

Csound (OSCsine.orc)

```
nchnls = 2
```

```
FLpanel "Panel1",100,450, 100, 100, 4  
gk1,iha FLslider "FLslider 1", 440, 220, 0 ,6 , -1, 15, 300, 40,50  
FLpanelEnd
```

```
FLrun
```

```
gihandle OSCinit 7770
```

```
instr 1  
kf1 init 0  
khandle OSClisten gihandle, "/mysynth/freq", "f", kf1  
OSCsend gk1, "localhost", 7770, "/mysynth/freq", "f", gk1  
a1 oscil 10000, kf1, 1  
outs a1,a1  
endin
```

Csound (OSCsine.sco)

```
f1 0 512 10 1  
i1 0 -1
```

Supercollider (OSCsine.sc)

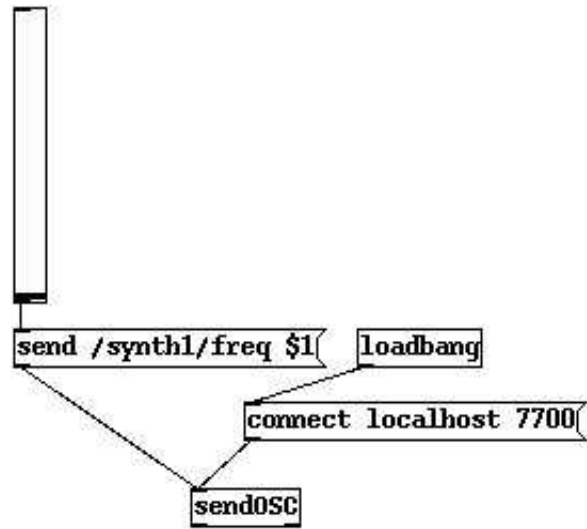
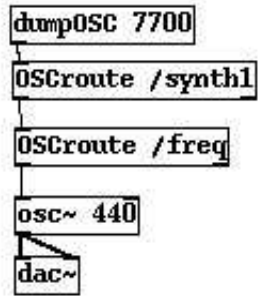
```
(
  SynthDef(\sinesynth,
    {
      arg freq = 440;
      Out.ar([0,1],SinOsc.ar(freq, mul: 0.1));
    }
  ).send(s)
)
(
  var node = Synth(\sinesynth).nodeID;

  OSCresponder(nil, '/synth1/freq',
    {arg time, resp, val; s.sendMessage("/n_set", node, "freq", val[1])}).add;

  n = NetAddr("127.0.0.1", 57120);
  w = JSCWindow.new("Slider", Rect(50,50, 40, 400));
  w.front;

  b = JCSlider(w,Rect(10,10,20,340));
  b.action = { arg butt; n.sendMessage("/synth1/freq", 220 + (220 * butt.value));
  };
)
```

PD (OSCsine.pd)



Appendix 3 - Delay

Csound (delay.orc)

sr = 44100

kr = 22050

ksmps = 2

nchnls = 2

FLpanel "Delay",200,450, 100, 100, 4

ihv1 FLvalue "secs", 50, 20, 45, 370

gk1,ihs1 FLslider "Time", 2, 0, 0 ,6 , ihv1, 15, 300, 60,50

ihv2 FLvalue "%", 50, 20, 105, 370

gk2,ihs2 FLslider "Feedback", 100, 0, 0 ,6 , ihv2, 15, 300, 120,50

FLpanelEnd

FLrun

instr 1

a1 in

adel delayr 2

atap deltapi gk1

 delayw a1 + (atap * gk2 * 0.01)

 outs atap + a1, atap + a1

endin

Csound (delay.sco)

i1 0 -1

Supercollider (delay.sc)

```
(
  var node;
  SynthDef(\del2,
    {
      arg dt = 1,
          fb = 0.1;
      var abuf, ain;
      abuf = LocalIn.ar(1);
      ain = AudioIn.ar(1);
      abuf = DelayL.ar(abuf, 2, dt);
      abuf = abuf + ain;
      LocalOut.ar(abuf * fb);
      Out.ar(0, abuf);
    }
  ).send(s);

  node = Synth(\del2).nodeID;

  w = JSCWindow.new("Delay", Rect(50,50, 400, 400));
  w.front;
  w.view.decorator = FlowLayout(w.view.bounds);
  w.view.decorator.nextLine;

  JEZSlider(w, 300 @ 24, "Time", ControlSpec(0, 2, \lin, 0.01, 1),
    {|ez| s.sendMessage("/n_set", node, "dt", ez.value); });

  w.view.decorator.nextLine;
  JEZSlider(w, 300 @ 24, "Feedback", ControlSpec(0, 1, \lin, 0.01, 0.1),
    {|ez| s.sendMessage("/n_set", node, "fb", ez.value); });
)
```

PD (delay.pd)

