Max/MSP exercises 4b

Ex.1

The last set of exercises explored the [sfplay~] object. This reads sound directly from the hard drive, which is useful for many purposes. But sounds can also played back from RAM (Random Access Memory). The advantage with this is that access to the data is considerably quicker.
To load data into RAM, we use the [buffer~] object.

1. Copy this object. Lock the patch and double-click on it: buffer~ mysound1 2000 2

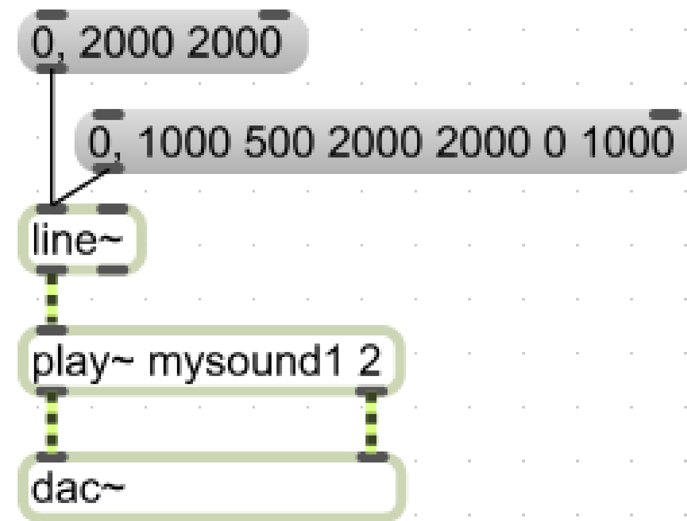
The window that pops up represents the available space in memory (2000ms in 2 channels) that [buffer~] has set aside in memory. We have given this space a name: 'mysound1'. This is important! Other objects will refer to this [buffer~] by this name in order to write to it, or to access its contents.

2. Copy the routine on the right. Lock the patch and double-click the [adc~] object to access the DSP window. Choose your computer's local driver (CoreAudio Built-In on the Mac) as we need access to the microphone and switch on the DSP. Press the [toggle] and make noises into the mic. You should see [buffer~]'s window update...

Ex.2

Okay, so you can write material into a [buffer~]. You can also read from it. There are a variety of objects that will enable you to do this. We will cover two of them.

1. Copy this routine:



2. Lock the patch and hit the two message boxes. You will notice that in this patch, [line~] is being used to read through the contents of the [buffer~] via the [play~] object. Depending on the values given (i.e. where it is reading to and how long it is given to get there), the sound will play back at different rates and can play backwards.

Ex.2 (cont)

This means we can use the [play~] object to generate some interesting sounds by 'scrubbing' backwards and forwards through a [buffer~] (as though moving tape over an analogue tape head).

3. Use a [function~] object to read backwards and forwards through the contents of [buffer~] over a period of ten seconds (you will need to change the *Lo and High Display Range* and *Hi Domain Display Value* within the [function]'s Inspector Window).

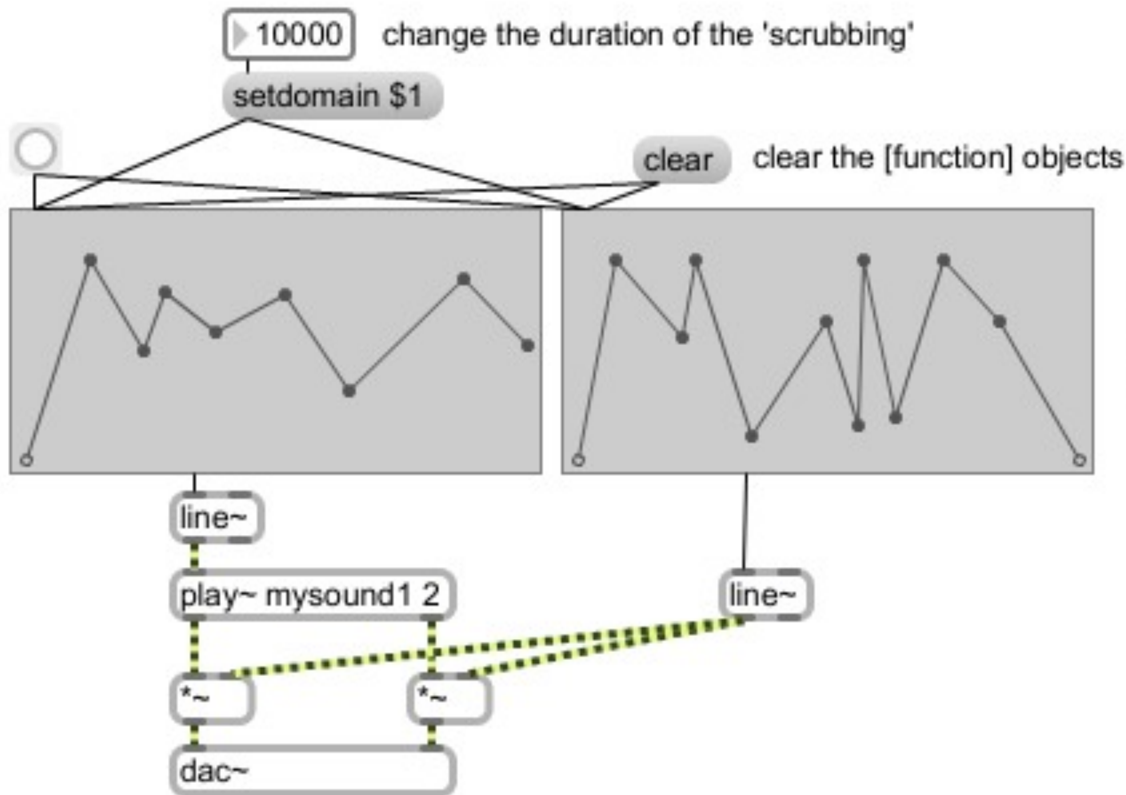
4. Add a further [function~] object that allows you to shape the volume of your 'scrubbing' over the same period.

5. How would you now reduce the duration of playback dynamically (rather than having to use the Inspector window)? [clue: alt-click on the [function] object to access its help page, then investigate the 'setdomain' message]

6. How would you get this scrubbing pattern to 'loop'? [clue: check the right-hand outlet of the [line~] object]

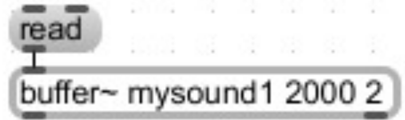
Ex.2 (cont)

Solution to Ex.2.3-2.6:



Ex.3

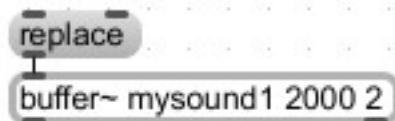
[buffer~] allows you not only to record sounds into it, but to import sounds from your hard drive.



1. Copy the above routine, lock the patch and click on the 'read' [message] box. Choose a sound you'd like to load using the dialogue that pops up. Double click on the [buffer~] object. You will see that the first two seconds of your file have been loaded.

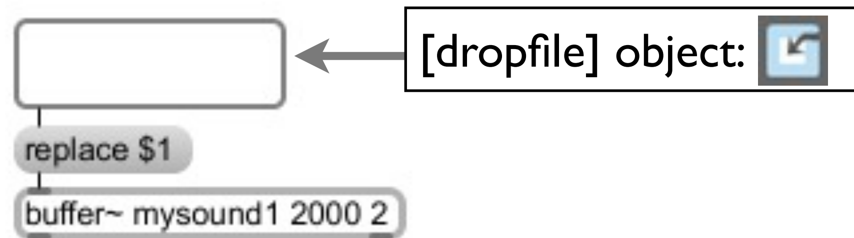
N.B. Your file on the disk was referred to only to retrieve the first two seconds'-worth of data (88200 samples if recorded at 44100 sample rate) and copy it into the [buffer~]. Any playback object that refers to [buffer~] will refer to *its* contents, *not* to the original sound file.

2. Modify the routine, replacing the 'read' [message] box with 'replace' and click on it. Check the contents of the [buffer~] object again. You will notice that the *entire* file now has been copied into the [buffer~].



Ex.3

3. Modify the routine again as follows:



This alteration allows you to load files simply by drag'n'dropping them into Max. Note the use of the $1 'wildcard' message again. You can probably see that this wildcard can be very useful in a variety of circumstances.

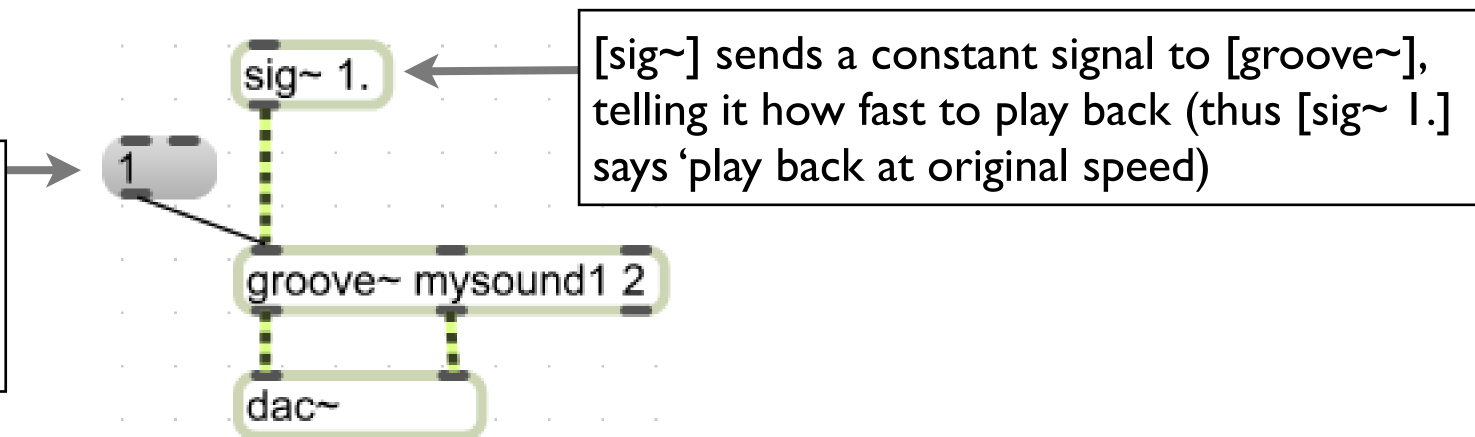
(You can, of course, use the [dropfile] object in conjunction with an 'open $1' [message] to load files into [sfplay~]).

Ex.5

A little more versatile than [play~] is the [groove~] object.

1. Copy this routine:

a message tells [groove~] where to start from. This says to start reading at 1ms.



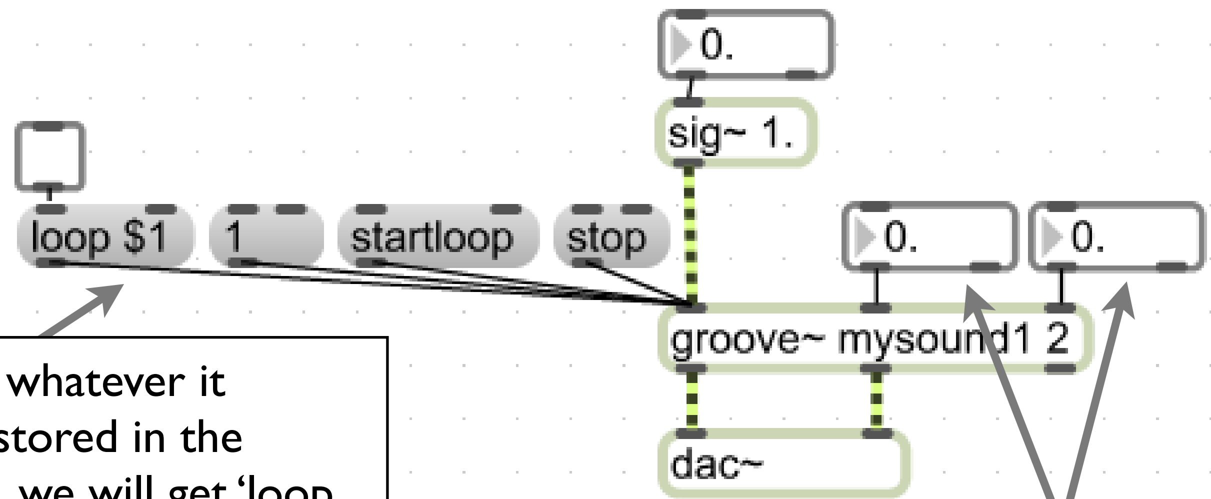
2. Lock the patch and click the '1' [message box]. [groove~] will play back the sound from 1ms to the end of the file at 1 x speed.

3. Change the value in the [message box] to '1000' and hit it.

4. Connect a [float] box to the inlet of [sig~], then lock the patch and change its value. You'll hear that the transposition behaves in the same way as it did with [sfplay~].

Ex.5 (cont)

5. Modify the patch as follows:



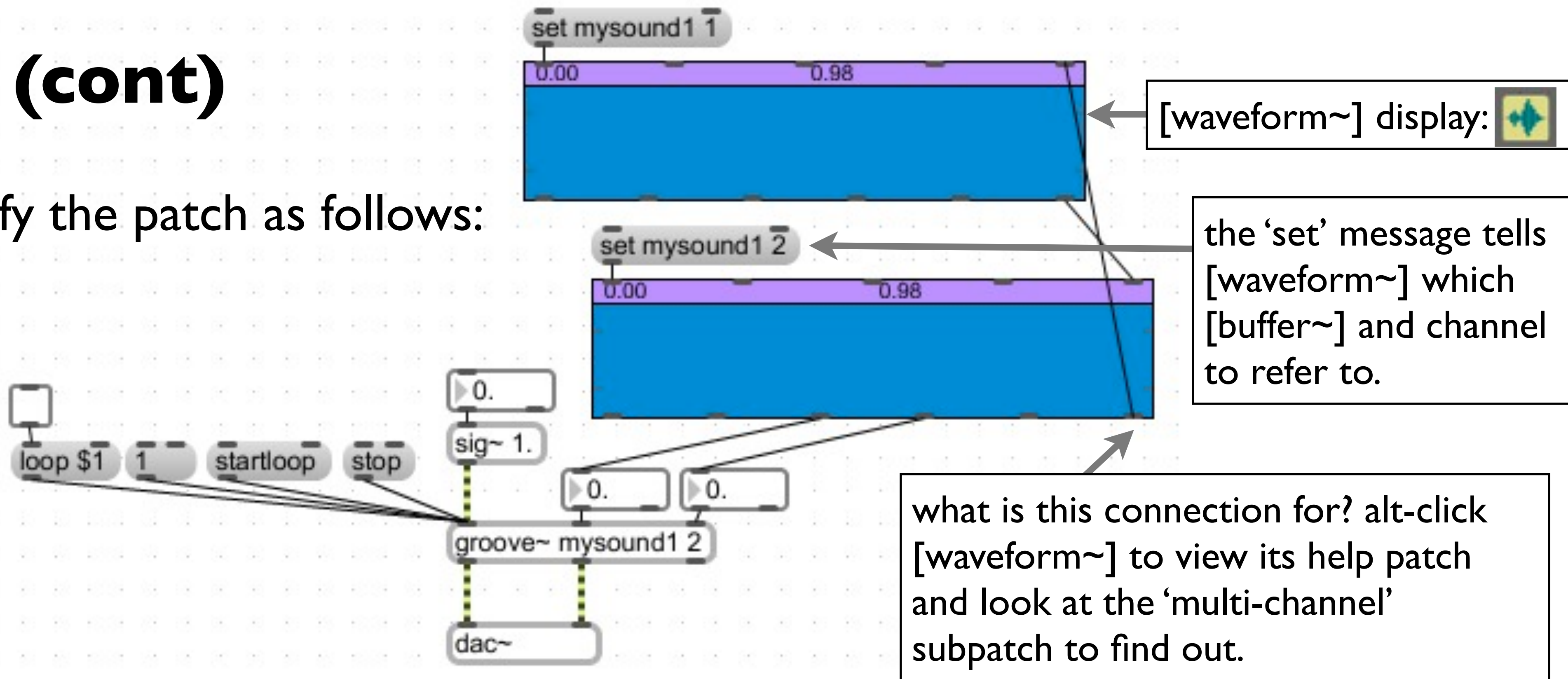
As before, the $1 'wildcard' throughputs whatever it receives, adding it to whatever's already stored in the [message box]. Here, if the [toggle] is on, we will get 'loop 1' (thus turning looping on); if it is off, we will get 'loop 0' (thus turning looping off).

loop start and end boxes. The left box (start) should be a smaller value than the right (end).

6. Turn looping on by clicking the [toggle] above the 'loop $1' [message box]. Then, set the values in the [float] boxes attached to [groove~] to '500' (left) and 1500 (right). Now hit first the '1' message box (wait to see what happens), then 'stop', then 'startloop'. You should notice that 'startloop' only plays the loop.

Ex.5 (cont)

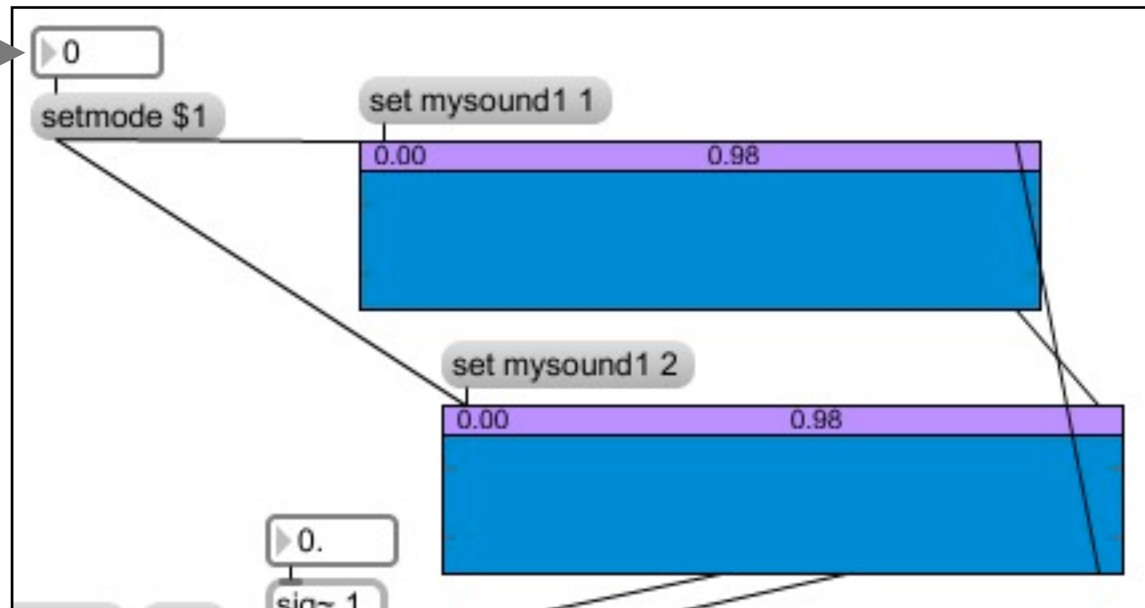
7. Modify the patch as follows:



8. Open the [waveform~] object's help patch (alt-click the object). From here, open the reference documentation for this object (click 'open waveform~ reference' at the top of the window). Read the entry (some way down the page) for 'setmode'.
Try implementing the various various view modes in [waveform~] and see what they do.

Ex.5 (cont)

0 = no mouse interaction
1 = select mode
2 = loop mode
3 = move mode
4 = draw mode



9. Explore these various modes with respect to how the cursor behaves when in the [waveform~] object. You should find that you can now define and move the loop points the [groove~] object.

Ex.6 CHALLENGE!

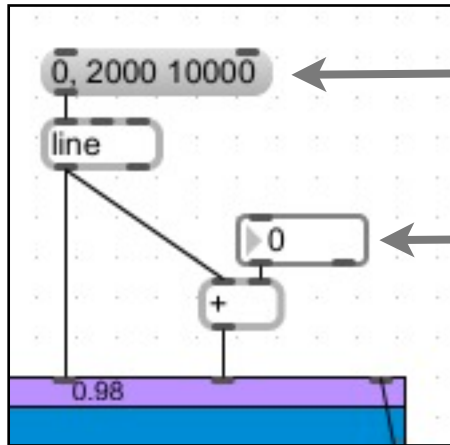
Now that you can move the loop points around within the [waveform~] editor, you can see that you can get some quite interesting granular sonic results from doing so.

1. Explore the inlets of the [waveform~] object. You will see that there are two that define start and end loop points.

2. Using a [line] and a [+] object, effect a granular timestretch over your soundfile. Consider:

- you will need your loop start point to move smoothly through time (use the [line] object
- you will need your loop end point to move smoothly through time at a consistent rate *relative* to the loop start point (to give you a consistent-length grain).

Ex.6 SOLUTION



(this assumes that your [buffer~] length is 2 seconds and that you want your timestretched result to be 10 seconds (your version might, of course, be different)

define grain length