Building Loopers in MSP

Live looping is rapidly becoming a musical genre unto itself. It imposes strict limitations on the form a piece must take, but still leaves a skilled performer wide leeway to create a unique compositions. One source of limitations is the hardware or software used. All a performer has available are the tracks, effects and controls the designer choose to put in. You can escape these limitations by building your own looper in MSP.

Basic Looper

Loopers are built on buffer~, which provides a section of memory to record audio. Figure 1 illustrates the simplest approach.

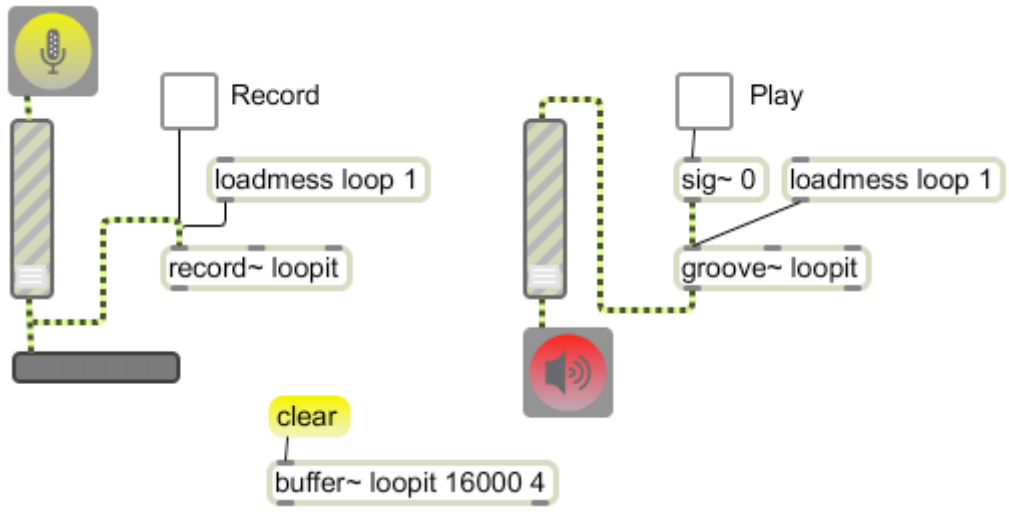

Figure 1.

The record~ object stores audio into the buffer~ object with the same name. Both the signal to be recorded and the 1 or 0 that control recording are applied to the left inlet. The other inlets can set points to begin or end recording. Recording will always begin at the start point when a 1 is received. If the loop 1 message has been received, recording will continue at the beginning after the end point is reached. Note that the patch includes a gain~ object and meter~ to ensure recording at the proper level.

The groove~ object plays audio from the buffer~ of the same name. Playback rate is controlled by the value of a signal applied to the left inlet. (Numbers at the left inlet cue playback within the buffer.) The other inlets set begin and end points. If the loop 1 message has been received, playback will be continuous.

Multitrack Loops

The patch of figure 1 lacks most features needed for live looping. The most vital is multitrack operation. A buffer~ can contain 2 or 4 tracks, but record always records all

or nothing. We need the ability to record into individual tracks and play those tracks in tight synchronization.

The first step is to get groove~ to control playback of two buffers~. We might assume that if we just sent the same commands to two grooves with the same sized buffers, everything would work out, but when we try, the two eventually go out of sync. To guarantee togetherness, we use the sync output of groove~ to control a play~ object.

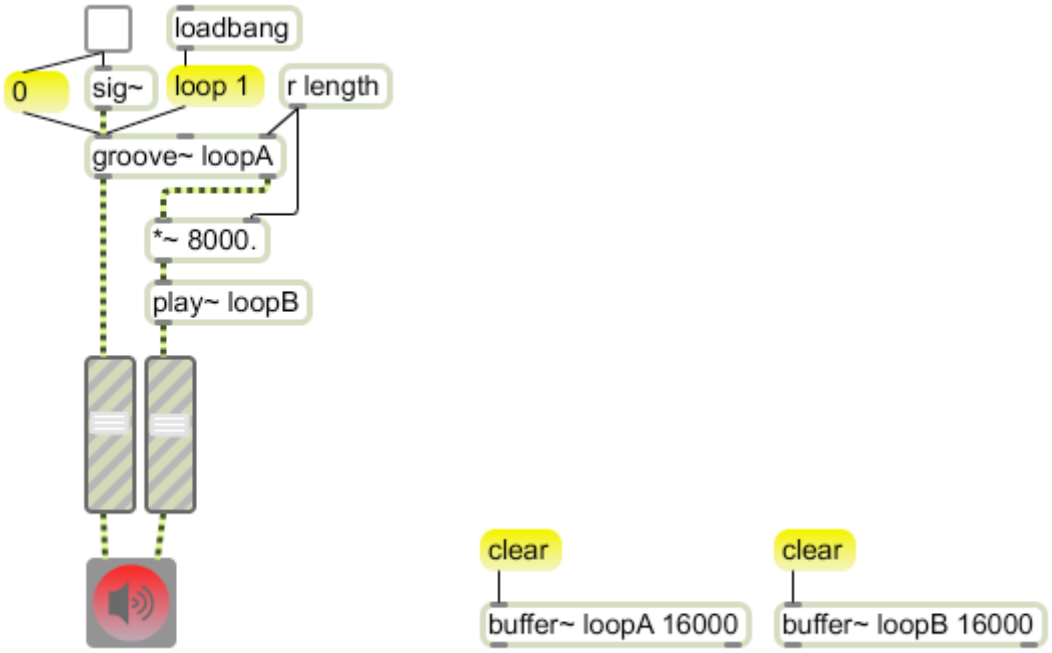

Figure 2
The right outlet produces a signal that varies with the playback position in the buffer. The range of this is 0.0 to 1.0, regardless of the size of the buffer. That signal is multiplied by the length of the loop (in ms) and sent to the play~ object linked to the other buffer. Now the two buffers will play together. Notice the end of the groove loop is set by a message sent to receive length. We shall see the source of this message soon.

The next step is to record to the buffers. Figure 3 shows the basic principle. You can change the target of a record~ object with the set message. Clicking the loopA or loopB message will choose the desired buffer to record. The length value must be reentered whenever the buffer is changed. This is managed by the trigger (t) object, which sends a bang after passing the name of the new buffer.

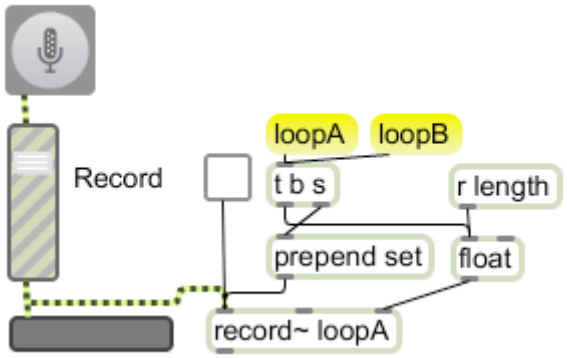

Figure 3.

This approach works, but is very hard to use. The problem lies in the way record behaves-- it always begins recording at the start of the buffer and stops at the end[1]. Getting a clean start to a recording is awkward at best. Further, playback with groove~ is entirely independent. If you are trying to match something that is already recorded, there must be a synchronization mechanism.

Clean starts are much easier if there is a metronome to follow, especially if the metronome provides a precount. Figure 4 adds a metronome to the patch.

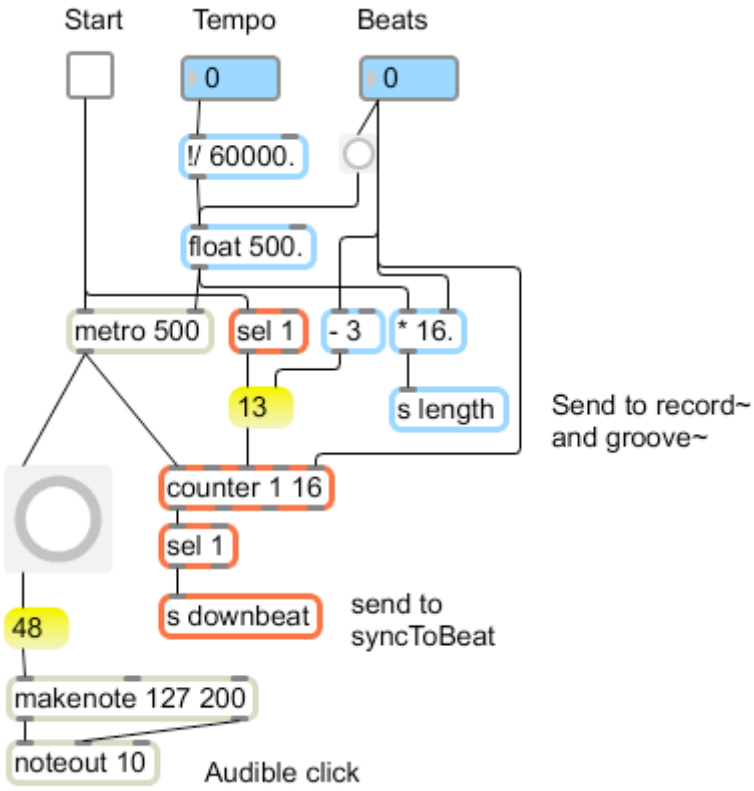

Figure 4.

[1] This behavior can be changed with the loop or append modes, but other problems turn up.

The uncolored section of figure 4 is a familiar patch that flashes a button and plays a note on channel 10. The blue objects set the tempo and number of beats. The interval for the metro object is found by dividing the tempo into 60000, and this is multiplied by the number of beats to set the total recording length.

The objects in red produce a bang on each downbeat. The first downbeat will occur 4 beats after the metro starts, because the counter will begin with count 13. There are some subtleties to the use of counter here-- since counter includes the length count, it must begin on 1 when it does a full cycle or on length - 3 to give the correct precount. The 1 is detected for the downbeat.

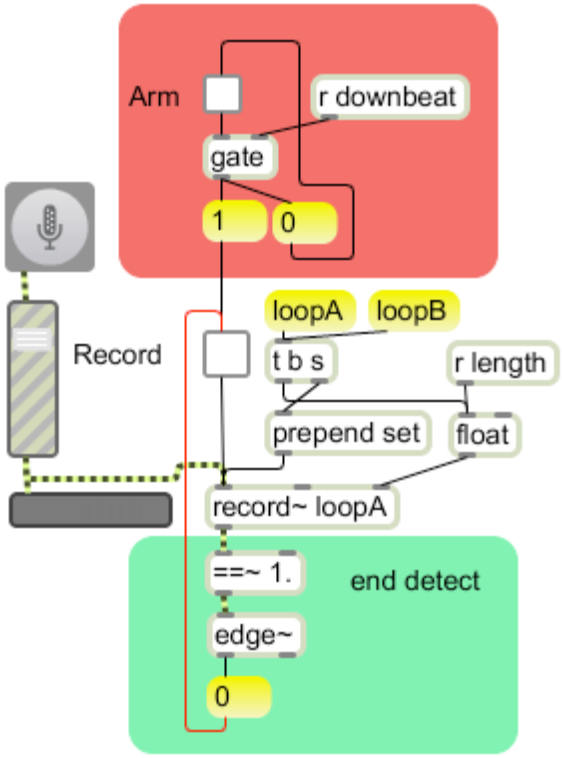

Figure 5.

To synchronize recording to the metronome, we make the additions shown in figure 5. The end detect section (green) follows the sync output of record~, which ramps from 0 to 1 as the buffer is filled. The 0 message does not stop recording, it indicates recording has stopped by clearing the toggle. Recording will always stop at the loop end.

The arming mechanism (red) contains a patch element I call a *mousetrap*. If the toggle is set, the gate is open, and will allow the next downbeat bang through to start recording. This same bang sends a 0 back to the arm toggle, so no further downbeats will be let through. Once this is place, we arm the toggle and start the metronome. Recording begins on the next downbeat.

Once a track is recorded, we change buffers. Now the goal is to begin recording in sync with the playback from groove~. This can be easily added with the same types of

mechanisms already described. We detect the end of groove~ and send a bang to a receive object called rollover. This is caught by the record patch in the same way the downbeat is.

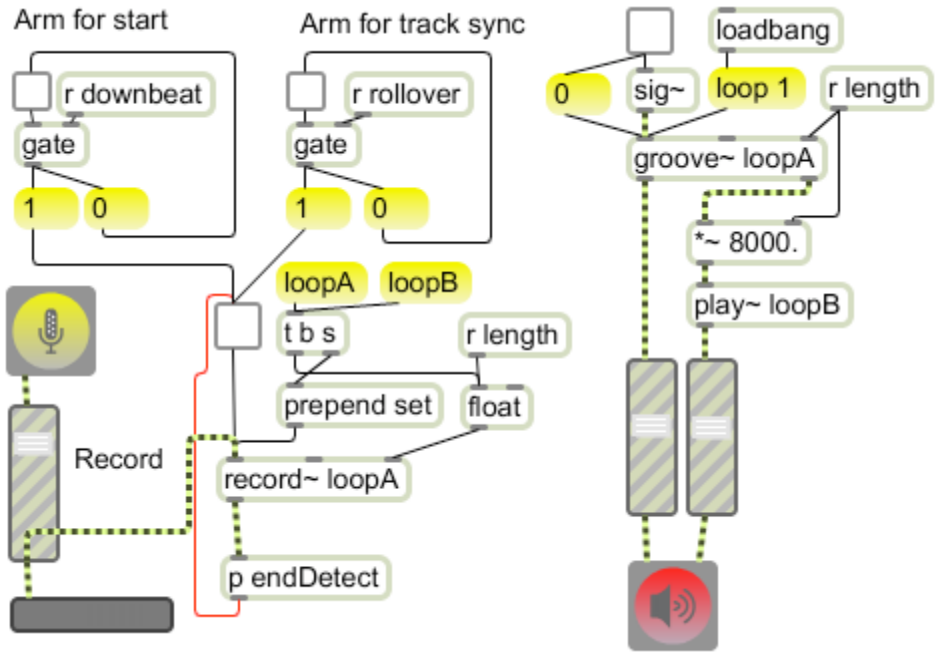

Figure 6.

To use the patch, set the arm for start toggle, then start the metronome. Recording will begin on the fifth click and end when the buffer loopA is full. To add a track, click the loopB message, then begin playing the loop. Some time before the loop repeats, hit the arm for track sync toggle, and recording will begin in loopB on the next pass. It's easy to add more tracks- all that is needed is another buffer~ and play~ object for each.

This patch works reasonably well, but does not match the flexibility of a commercial loop box. The following features are missing:

- Arbitrary recording start points
- Recording triggered by audio input
- Length set by punch out

It turns out that record~ does not work well for arbitrarily punching in a loop. If loop mode is off, recording ends at the end of the buffer. If loop mode is on, recording can continue past the end, but it loops back to the start point, not the beginning of the buffer. Punching in at an arbitrary point and recording the entire buffer is not available.

Looping the Hard Way

To get the features we want, we need to build the looper out of more primitive objects. For recording, we use a 4 track buffer~ and poke~, as shown in figure 7.

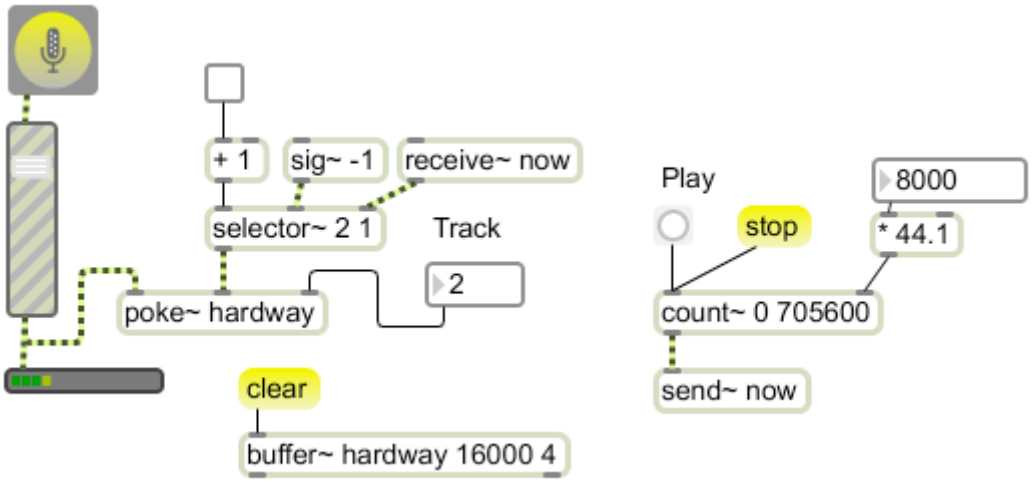

Figure 7.

Poke~ writes samples directly to the associated buffer~ at the address specified by a signal in the second inlet. An address signal contains the sample numbers in the buffer~ to replace. For a 16 second buffer~, the numbers run from 0 to 705,600 (at a 44.1 kHz sample rate). This signal can be handled by all of the MSP mechanisms, but you wouldn't want to listen to it. Poke~ can record to any track of a buffer~. Tracks are chosen by a 1 to 4 at the right inlet.

It would be nice if we could get the address signal directly from groove~ in the playback section, but the formats are incompatible. Converting from the 0 to 1 sync of groove~ to sample number introduces address errors and distortion. The only high quality address generator is count~, and we will use that as the master for both recording and playback. Count requires a bang to start and the message *stop* to stop[2].

The addresses produced by count~ will wind up at the [receive~ now] object. The selector~ chooses between that and a signal of -1, which inhibits recording. Recording is enabled with a toggle. Adding one to this makes selector~ choose -1 or the address signal.

Playback via count~ uses the index~ object. Index~ is the opposite of poke~. Give it addresses, and the values are output. Index~ is illustrated in figure 8. Note that an index~ object is required for each track of the buffer~.

Figures 7 and 8 are the basic record and playback mechanism. The essential difference between this and the approach in figure 6 is that recording can begin at any point in the loops. The same sort of synchronization can be applied, based on a metronome and detection of the end and zero point of the address counter[3].

[2] Count~ will repeatedly put out the same value when stopped.
[3] The end point is one less than the number of samples in counter. When using ==~ to find a spot in the count, it is important to specify the spot as an int.

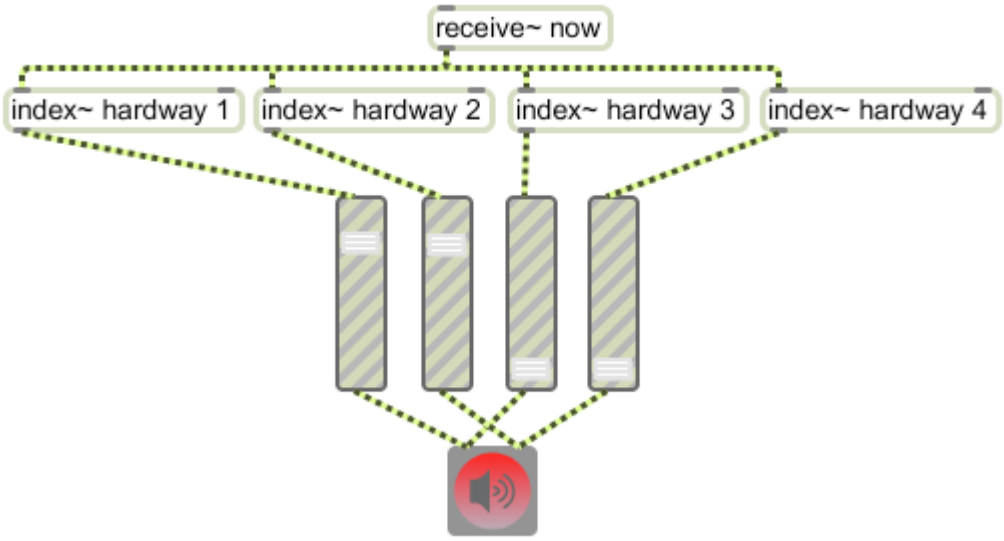

Figure 8.

This patch can be thought of as an endless loop of tape in which any section of the four tracks can be recorded. There's really no point in worrying about the numerical start and end points, but we do want convenient control of the process. One popular approach is to trigger recording when input begins and end with the spacebar. If we are recording on track 1, the recording time should set the length of the loop. We need to generate signals to control four actions:

1. Start the counter running from 0
2. Begin recording
3. Stop recording
4. Adjust loop length.

Actions 1 and 2 are triggered by sound input, the others by the spacebar. Figure 9 shows how to generate the control messages to start and stop recording.

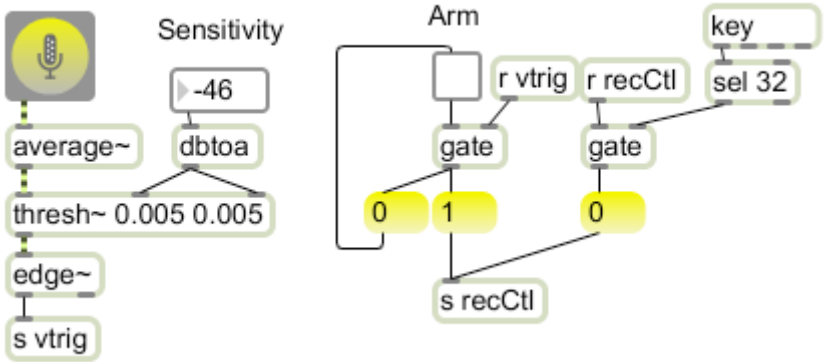

Figure 9.

The objects attached to the input will generate a bang when the input signal passes a threshold amplitude. Average~ measures the level, the thresh~ output changes to 1 when the threshold is exceeded, and the edge~ object produces a bang. The bang is sent as vtrig.

Vtrig is applied to a mousetrap mechanism. If the arm toggle is set, a 1 is sent to recCtl and the toggle is turned off. RecCtl is picked up several places, including a gate that allows a bang from the spacebar (the key and sel objects) to turn recCtl off. Figure 10 shows how these are used in the recording patch.

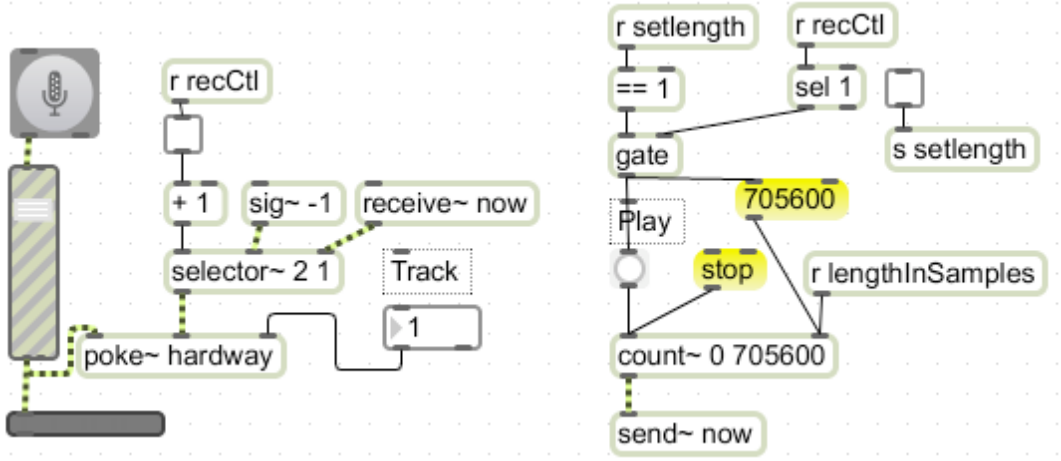

Figure 10.

RecCtl is used to enable recording. If you want this pass to set the length of the loop, turn on the toggle connected to send setlength. If setlength is 1, recCtl will also start the count~ object from 0. When the recording on track 1 is complete, the mechanism in figure 11 will set the length of the count.

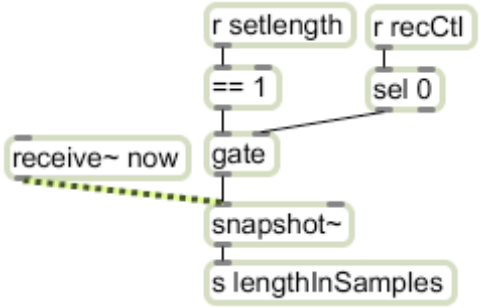

Figure 11.

This mechanism is a little harder to assemble than the basic record~ and groove~ patch, but it can be used as the foundation for elaborate and flexible recording systems. Potential additions might include:

- Better control layout (probably a presentation).
- A different way to indicate when you are recording the master track.
- More tracks with a second buffer~.
- A fadeout to remove the pop that occurs when the counter stops.
- Hardware control.

Time Stretching Loops

One popular effect for live looping is the time stretch. This is produced by simultaneously slowing or speeding playback and applying a compensating pitch change. The ratios are easy to calculate-- the pitch change is the inverse of the rate change- if you halve playback speed, double the pitch. Figure 12 shows how to do this in Max.

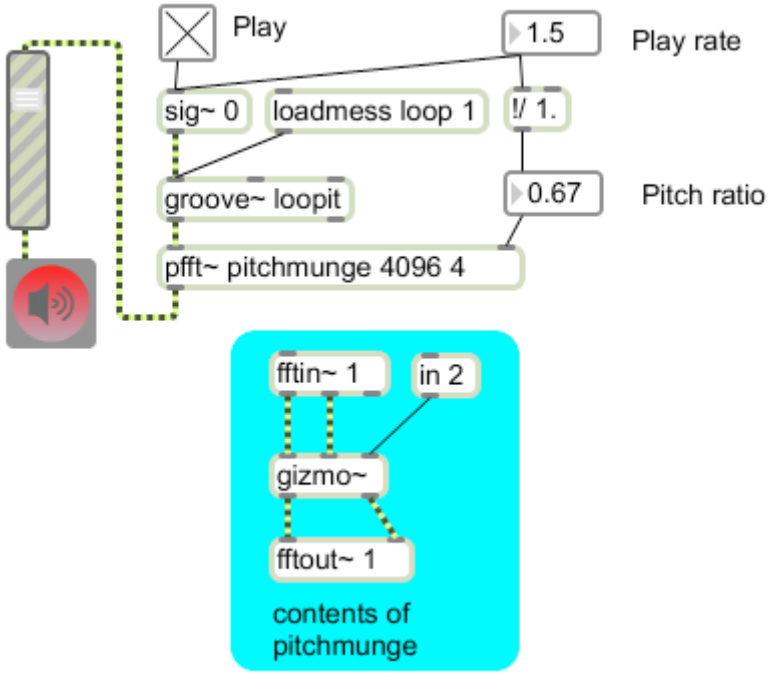

Figure 12.

Pitch changes are produced by the gizmo~ object within a pfft~ object. (Details are discussed in the Fourier Notes tutorial.) Since a groove~ can refer to any buffer~, this can be added to any of the patches discussed above.

Open Loops

The live looping techniques shown above use what I call closed loops, short patterns that are captured and repeated in various mixes. An even older looping technique is based on open loops. These were originally produced by stringing tape between two tape decks, one recording and one playing back, a system made famous by Pauline Oliveros' "Bye Bye Butterfly" and Terry Riley's "Come Out". A tape delay processor known as the echoplex used to be a staple of guitar players like Les Paul. The Max version is done with tapin~ and tapout~ as shown in figure 13.

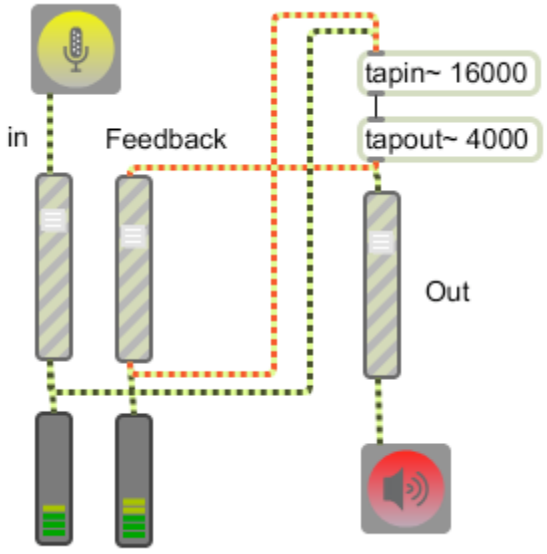

Figure 13.
This patch should win an award for the most fun you can have with the fewest objects. In fact, I routinely use it in first day demonstrations of Max. The output of the tapout~ is fed back to the tapin~ via the red patch cords. Whatever you put into it will come back again and again and again, fading a bit each time. You have to watch the feedback level carefully, as it does tend to either overload or loose the built up pattern. This is not a difficult skill to learn, but you can make a patch that will run unattended by putting a compressor in the feedback path as shown in figure 14.

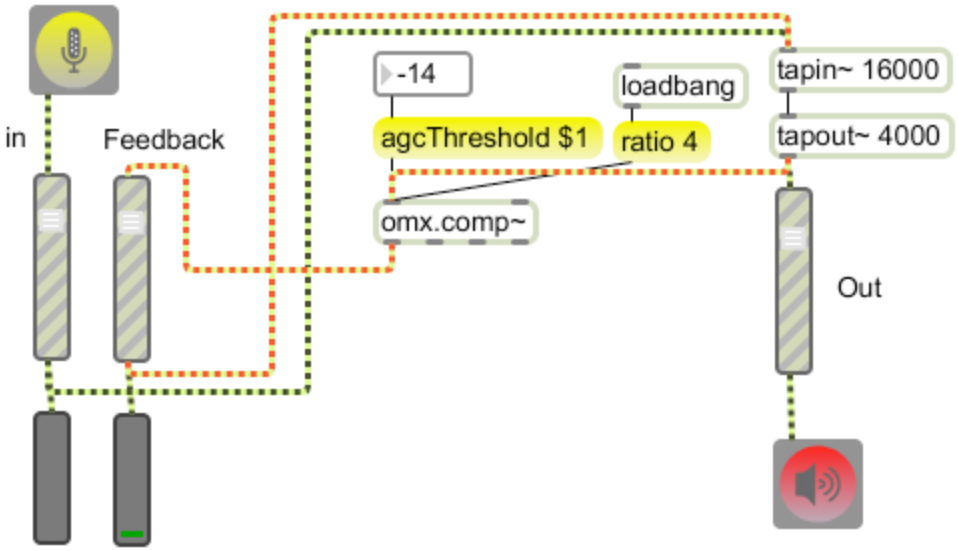

Figure 14.

All kinds of effects can be put in the feedback loop with fascinating results. One of my favorites is a 1/3 octave filter as shown in figure 15. This produces a classic effect known

as "screen" -- every time a sound cycles through the loop, it is changed by the filter until it approaches a pure tone.

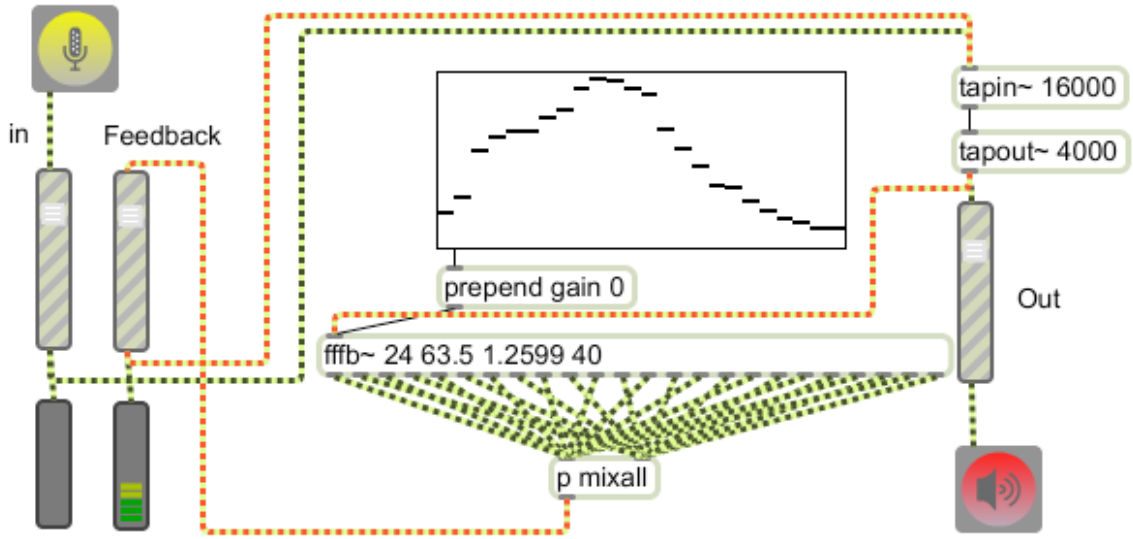

Figure 15.
The pitch changer from figure 12 can produce some wild effects and pretty sonorities. Note that in figure 16 the pitch change is specified in semitones. Try with a mix of short and long delays.

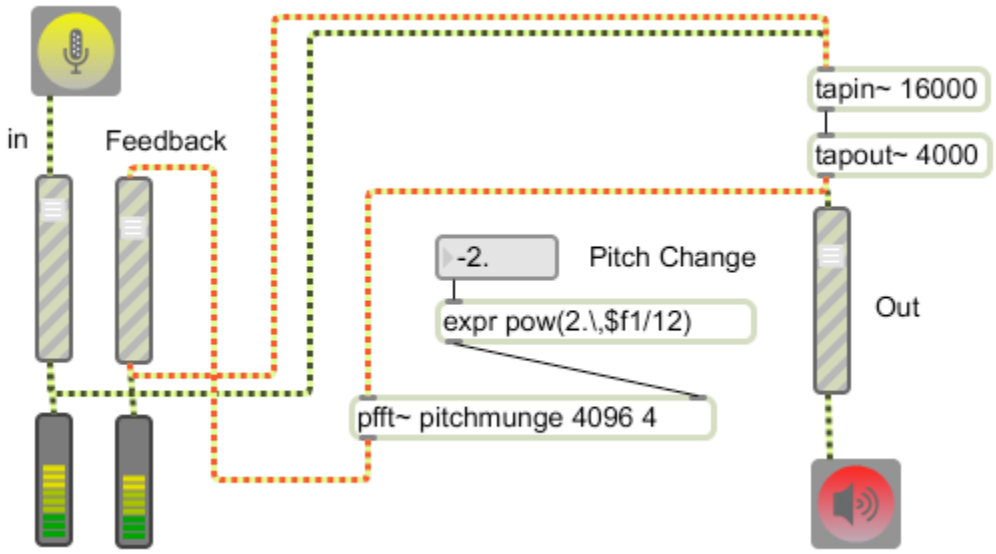

Figure 16.

The open loop technique can be further expanded by more delays and multiple effects. The loops can be quite long-- in fact it is quite effective if a sound comes back after it has been nearly forgotten. The most complex pieces are scored, letting a performer play a canon with himself.