Andrew Langdon
arl2178
Music Signal Processing
Final Project

After creating a basic flanger in Pd for my class presentation, I became more interested in understanding how the classic "audio effects" work on a more detailed level. Playing with a guitar and some effects pedals gave me a feel for how the effect changes the output audio, but it wasn't until I took Signals and Systems that I could begin to understand the math behind the transformations. With the information I've learned from the Music Signal Processing, I wanted to implement these effects in Pd and encapsulate the logic into modular Pd patches for future use. My goal was to create a group of patches for some of the effects that we didn't cover in class: echo, flanger, phaser, chorus, and a ring modulator.

Echo

The echo patch takes an input signal and outputs the same signal repeatedly with lower amplitude and a time delay. My patch creates a basic delay line using two delread~ and delwrite~ objects, one for the right channel and one for the left. The patch reads in audio from the inlet~, and pipes it to two delay lines, one for the right channel and one for the left. Then, the patch pulls a delayed audio signal from one channel's output and writes to the other's delayline, with a variable amount of feedback. This gives a mild ping-pong effect when the panning is moved away from the center value. The patch also contains a volume slider so that the volume of the echoes can be adjusted. Finally, the 'time' slider adjusts the delay time by giving an input to the variable delay-read object vd~.

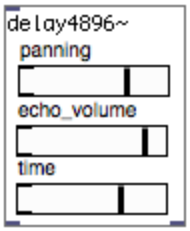

Figure 1: interface for basic delay4896~ patch

More than anything, making this patch taught me that keeping the internal circuitry clean becomes difficult very quickly, especially when the circuit requires feedback or when trying to map the controls of the interface to object inputs in the patch. Luckily, I remembered that wireless data-transfer was available and put it to use when making future patches. I also realized that the user has no way of controlling the sliders with other Pd objects, so any sort of custom modulation of the parameters is unavailable. In my next patch, I adjusted the sliders to be vertical and lined them up under inlets~. Designing echo served as an introduction to using

'graph-on-parent' and encapsulating the internals of a Pd patch into a useable interface.

Flanger

The flanger audio effect is created by mixing two identical audio signals together, time-shifting one back and forth by a few milliseconds. The phase-cancellation of the combined signal produces a comb-filter effect that sweeps over the frequency spectrum. I created the flanger Pd patch by writing the input signal to a delay line, and then reading from the delay line with a variable delay time using vd~.

Figure 2: the flanger4896~ object interface

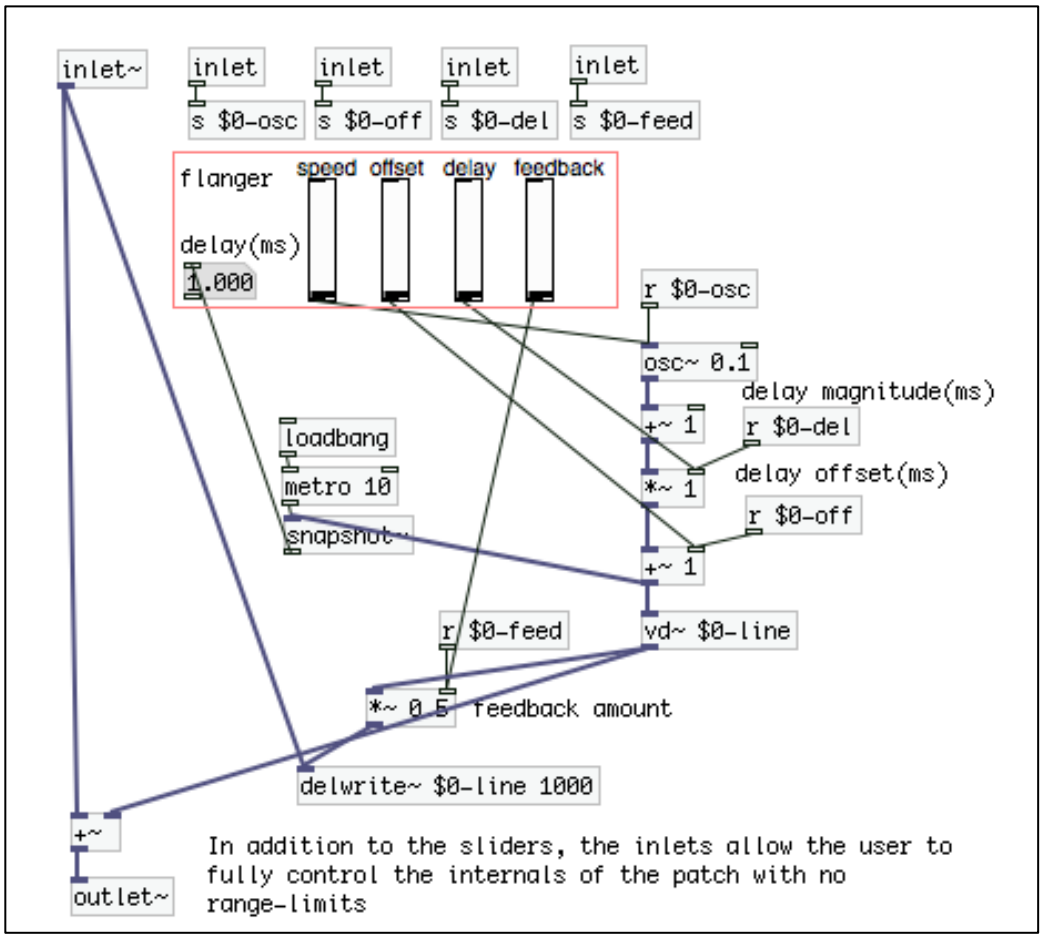

Figure 3: The internals of the flanger patch. The sliders control the flanging over a small range, while the inputs have no range restrictions except for the feedback, which has been range-limited in the updated version of the patch

Most of the controls for the patch dictate the way the delay-time changes. The output of an osc~ object is manipulated and fed in to the inlet of the vd~, and then the output of vd~ is sent to the patch outlet and also written back to the delay line. The 'speed' control determines the period of the time-shift of the delayed signal, and the range is from 0.001 Hz to 1Hz. The 'offset' slider determines the minimum delay time of the secondary signal. When the offset is 0ms, there is a moment when the secondary signal lines up with the original. When it's set at 5ms, the secondary signal is always at least 5ms behind the input. The 'delay' slider control controls the magnitude of the delay oscillator. The values for this slider range from 0ms to 20ms. When determining the ranges for the sliders, I found that most of the time I wanted delay ranges within 0ms-20ms for the standard flanging effect. Values in this range bring more character to the original input without changing the original signal too much. On the other hand, there are also times when using values out of this range is useful for the creation unusual and unique sounds. By adding inlets to the patch that correspond to the parameters of the flanger, the user can override the range restrictions on the patch and directly control the ranges and values that the flanger uses. A small number object shows the delay in milliseconds of the secondary signal so the user can get feedback while manipulating the controls.

Phaser

A phaser also modifies a signal using phase interference, much like the flanger. Instead of time-shifting the signal, the phaser uses a series of all-pass filters that modify the phase of the input signal. When the modified signal is recombined with the input signal, this creates a series of troughs and peaks in the magnitude of the frequency response. The math behind the manipulation is a little bit beyond my current knowledge, but I found some helpful information from Miller Puckette's Pd documentation at http://crca.ucsd.edu/~msp/techniques/v0.11/book-html/node161.html. By placing a zero and a pole in a specific way, our filter will not have an effect on the magnitude response of the frequency spectrum but will alter the phase of different frequencies. By using a low-frequency oscillator to control the filter coefficient, the phase response will fluctuate and when mixed with the original signal this will cause sweeping phase-cancellation across the frequency spectrum.

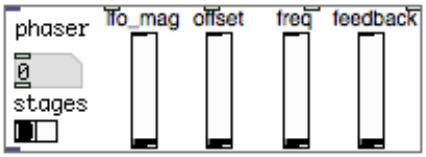

Figure 4: the phaser4896~ interface

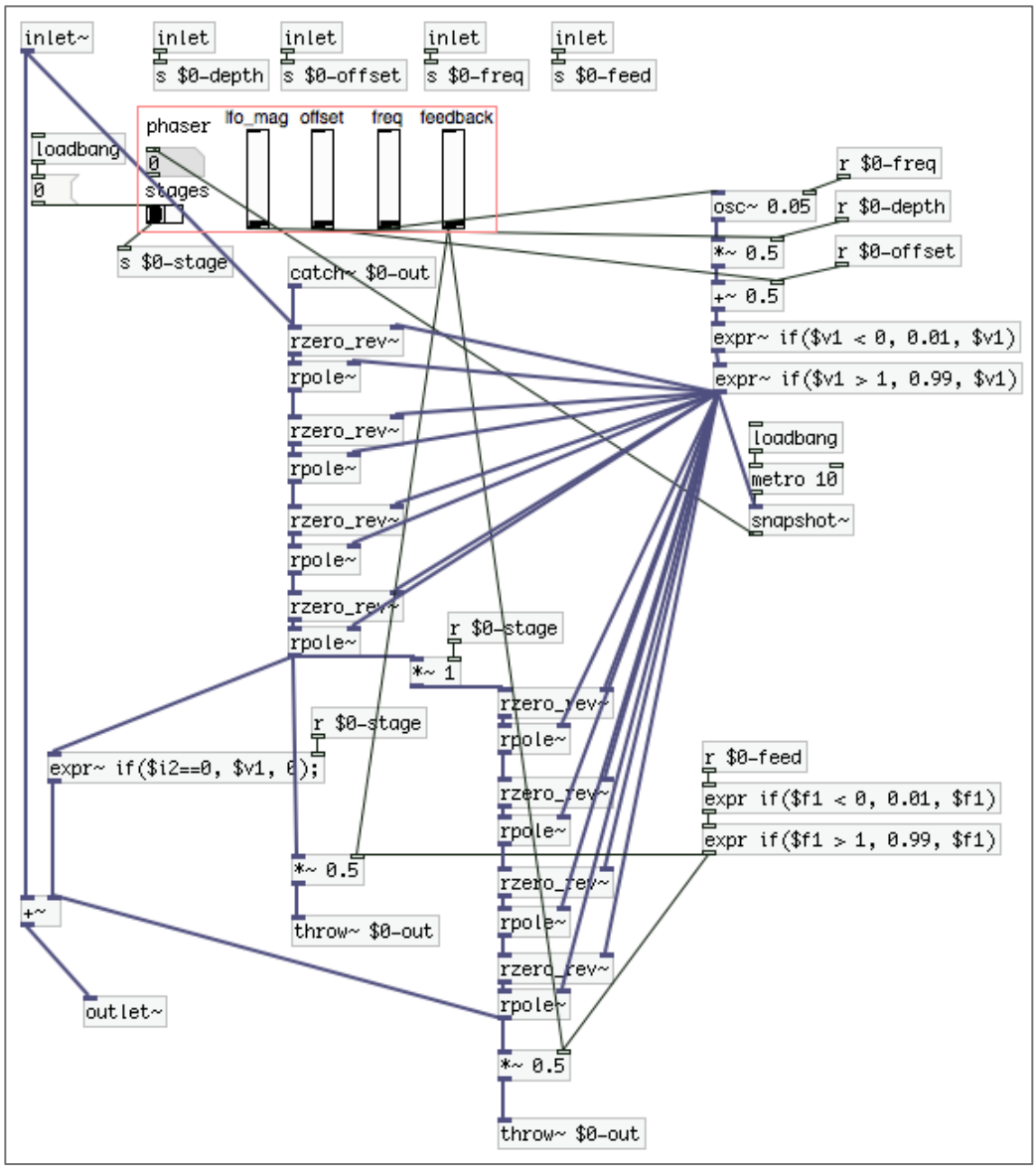

Figure 5: inside the phaser4896~ patch. The feedback expr object has been corrected in the actual patch

The phaser4896~ patch is a little more complicated than the flanger patch. The input enters the patch through the top-left inlet. The signal branches then branches, with one branch entering the all-pass filter stage. The rzero_rev~ and rpole~ objects in series create an all-pass filter, and the right inlets of these objects control the coefficient of the filter. An osc~ object controls the magnitude of the coefficient, and two expr~ object make sure the coefficient stays in the relevant range. After passing through the series of filters, the signal reaches another branch. If the first 'stages' radio button is selected, the filtered audio is fed back to the beginning of the filters and also combined with the unaltered input and sent to output. If the second 'stages' radio button is selected, the filtered signal is sent through another four filters with feedback, and then the filtered audio is combined with the original signal and sent to the output.

I learned very quickly that you must type carefully when dealing with feedback and filter coefficients. A misplaced decimal point has the ability to blow your eardrums out if you're using headphones. I wanted the user to be able to override the patch controls with their own inputs, but exposing the user to potentially ear-harming noises when they're experimenting with the patch seemed like a mistake. I chose to limit the ranges of the input by taking the values from the inlet and then limiting them to the relevant range by using expr objects that perform simple minimum and maximum calculations. This way, the user can feel safe hooking up wires to the inputs without having to check the internals of the patch.

The patch contains a few controls. A number object shows the coefficient of the filter so the user can get some feel for what the inlets and controls are doing. The magnitude control determines the range of coefficients that the LFO sweeps over. The offset control sets the center of the LFO. The frequency slider controls the speed of the oscillator, and the feedback slider changes the amount of the filtered signal that gets written back to the delay line.

Chorus

The chorus effect is created by making multiple copies of the original signal and pitch-modulating them with an LFO so that the output is perceived as being created by multiple sources. Usually, the output signal sounds wider and denser than the original. The patch I created writes the input signal to a delay line and then reads the signal back from the delay line with four different variable delays, controlled by four LFOs. Each LFO has a slightly different phase and speed as to not reinforce any phase cancellations, which would lead to a flanger-like sound. The overall output is spread over the stereo spectrum by using two outlets and two voices per channel, with each channel using different LFOs to control the delay line read.

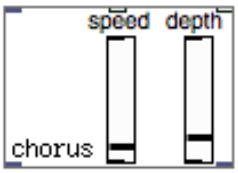

Figure 6: the chorus4896~ interface

The chorus interface is very basic. The 'speed' control determines the frequency of the voice-modulating LFO, and the depth control the amplitude. A higher amplitude gives the voices a larger detune from the original signal, and the speed determines how quickly the pitch fluctuates. The range for the speed slider is 0.01Hz to 5Hz, and the depth slider multiplies the delay-line lookup oscillator by up to 5 times.

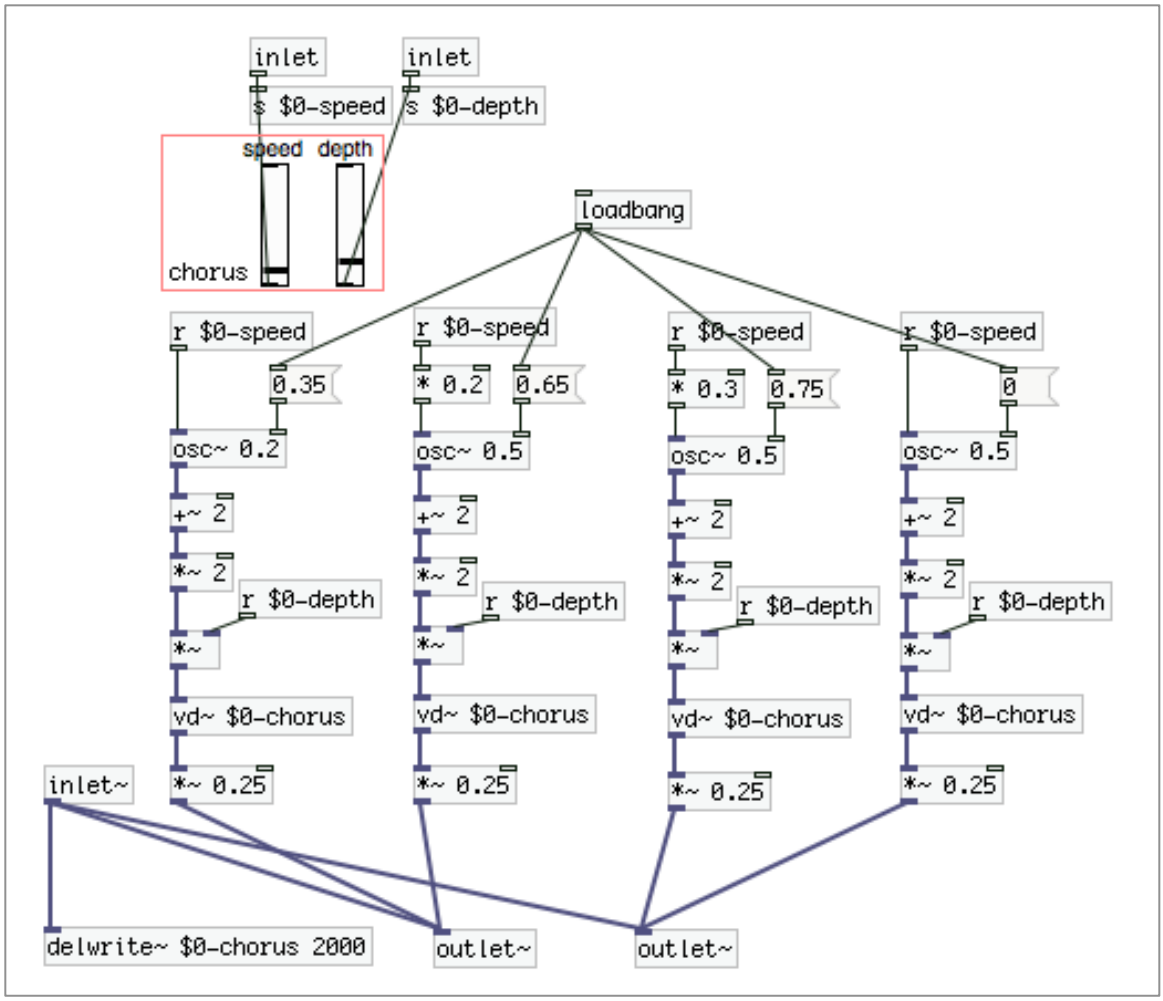

Figure 7: the internals of the chorus4896~ patch

Internally, four osc~ objects are shifted and scaled so that their outputs are in the positive range. Then, they are scaled again by the depth parameter, which can be overridden by attaching a float to the second inlet. The chorus outputs four voices in stereo, along with the original signal. Without feedback, the patch does not need guards against overflow, so the user has full control over the range of the controls via the inlets at the top.

Ring Modulator and Sample and Hold

Ring modulation involves taking an input signal and multiplying it by another signal to produce amplitude modulation. This produces a unique sound, sometimes creating partials that are harmonically related and at other times creating an unrelated set of partials. It can be used to create metallic, bell-like sounds if used with the proper parameters. In the ringmod4896~ patch, the input is split in to two tracks, and each is modulated with the output of an osc~ of varying frequency. One signal is sent to the left output, and the other is sent to the right. The sample and

hold section of the patch implements the samphold~ object built in to Pd. The sample rate control determines how often the input signal is sampled by controlling the frequency of a phaser~. When the phaser~ output drops back to its minimum level, the samphold~ object holds the input value until the phaser completes another period.

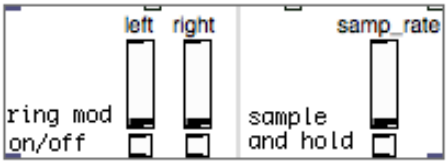

Figure 8: ring modulator interface

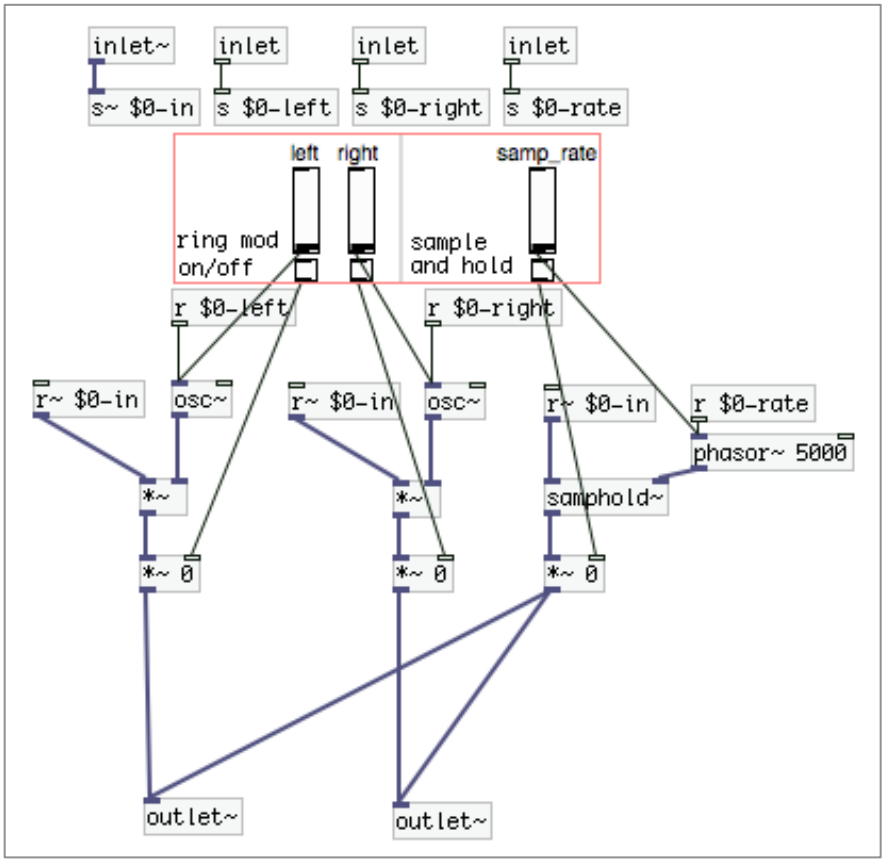

Figure 9: inside ringmod4896~

Although the ringmod4896~ object is one of the simpler objects in the set, I think it creates some of the more interesting sounds out of the whole effects suite. The output is fairly unpredictable, even though the manipulation is quite simple. It works well with a vocal sample as the input, and I've included a short sample in the same folder as this PDF.

Overall, programming in Pd has been an enjoyable experience except for the few times I mistyped the coefficients for the feedback loops. It's pretty clear that I've

only scratched the surface of what can be done with Pd and I look forward to using it more often in music and for my personal projects. It can be difficult to keep everything organized inside the patch, but I think this difficulty can be beneficial. To create an organized Pd patch you must first organize your thoughts. With a clean patch, it's easy to go back and make extensions and modifications.

Included with this PDF:

- delay4896~.pd
- flanger4896~.pd
- phaser4896~.pd
- chorus4896~.pd
- ringmod4896~.pd
- allpatches.pd, which contains all the patches pre-loaded
- VocalSample.wav, from Stanley Kubrick's 'Dr. Strangelove'
- sitar.wav, from a Ravi Shankar improvisation session
- The images from this PDF