THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

SCHOOL OF MUSIC

CHAINLINKFX: THE RASPBERRY PI AS A GUITAR EFFECTS PROCESSOR

BRIAN FAY
SPRING 2014

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Information Sciences and Technology
with honors in Music

Reviewed and approved* by the following:

Mark Ballora
Associate Professor of Music Technology
Thesis Supervisor, Honors Adviser

Paul Barsom
Associate Professor of Music Composition
Faculty Reader

* Signatures are on file in the Schreyer Honors College.

ABSTRACT

ChainLinkFX is an audio processor intended for use in guitar performance. It features two parallel chains of effects, which may be reordered and configured during performance by pressing buttons on a MIDI controller. The central element of the system is the Raspberry Pi, an affordable single-board computer. A QuNeo MIDI controller is connected to the Raspberry Pi, allowing effects to be switched by hand. Additionally, a momentary push button is used to implement foot control. Using open-source software to process audio, ChainLinkFX offers customization that traditional guitar effects pedals cannot allow.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank the many people who made this project possible. Mark Ballora and Paul Barsom oversaw the project, offering plenty of encouragement and advice along the way. I have absorbed tons of information from them over the past few years; their courses are easily some of the best I have taken at Penn State. I also want to thank Ann Clements, who met with me early in the planning stages of the project and helped convince me that it was worth pursuing. All of these professors have made me feel very welcome in the School of Music, which is a bit removed from my usual territory.

Dave Mudgett was very helpful in the nascent stages of the project, supporting the somewhat unorthodox idea of an IST student writing low-level audio processing code. He is living proof that programming, engineering, and guitar-playing are not mutually exclusive interests!

The community surrounding Pure Data has been an invaluable resource in this project. Who could have predicted in 1996 the amazing applications that would be possible with Miller Puckette's new programming language? This project also would not have reached fruition without the groundwork laid by Raspberry Pi enthusiasts and the smart people at Stanford's CCRMA.

Finally, I must thank my mother for introducing me to music, my father for showing me the wonders of technology, and my sister for inspiring me to strive for academic success. Attempting to read her thesis with no background in Biology is a humbling experience. My family has always supported me, and I can't imagine where I would be without their encouragement.

Chapter 1

Introduction

For years, homemade guitar effects pedals have been built by wiring together complicated electrical circuits. Some of these pedals use only analog components to process signals, while others incorporate microcontrollers for digital signal processing. Building these effects pedals requires experience with soldering and troubleshooting circuits. Most pedals are single-purpose, only featuring one type of effect. Changing the effect would require building an entirely different circuit.

Using a computer to process signals offers more flexibility – swapping an effect can be as simple as running a different program. Additionally, computers can be used to develop effects that would be extremely difficult to model with electrical components. Digital audio samples are stored in memory, allowing loopers and other delay-based effects to be constructed easily.

Despite their usefulness in signal processing, personal computers are only occasionally used as alternatives to guitar effects pedals. Laptops and desktop computers are more expensive and less portable than traditional effects pedals. ChainLinkFX aims to implement a custom effects pedal using an alternative device – the single-board computer.

Recently, single board computers have become powerful enough to perform real-time audio processing. These computers are tiny enough to fit inside small enclosures, and they are inexpensive when compared to laptops and desktop computers. ChainLinkFX is a guitar effects processor that uses one of these computers, the thirty-five dollar Raspberry Pi Model B (Raspberry Pi Foundation, 2014). The system sends a guitar signal through multiple parallel chains of effects, which may be tweaked and repositioned using a Keith McMillen Instruments

QuNeo MIDI controller (Keith McMillen Instruments, 2012). Using the Raspberry Pi to process audio helps address the issues of building custom guitar effects pedals.

Figure 1: ChainLinkFX – A custom guitar effects processor, built using a Raspberry Pi computer

Motivation

For an electric guitarist, sound is tied to signal. Processes that occur between the pickups and the amplifier have the power to redefine the character of the instrument. Although countless existing effects pedals already present musicians with an overwhelming amount of possible sounds, there is a certain appeal to developing one's own device. A company is restricted to making products that it can sell, but an individual can create something that is tailored to his or her own needs. For musicians who seek to develop their own distinctive voices, this individual freedom is a blessing.

Freedom to independently create things has fueled the development of “Maker” culture, whose participants use technology in creative do-it-yourself projects (Maker Media, 2013). One device that has been very popular to these developers is the Raspberry Pi computer. The compact size, affordable price tag, and variety of connections that are offered by the device make it attractive for many purposes, including musical applications.

The Raspberry Pi can connect to audio and MIDI devices via USB, and it can also interface with electronic components with its General Purpose Input Output (GPIO) pin. Additional electronic connections could be made by connecting to an Arduino board. The Raspberry Pi can be housed in a small container, like the metal enclosures used in many guitar effects pedals. If necessary, the computer is fairly inexpensive to replace. All of these things make the Raspberry Pi an attractive option for processing a guitar signal.

By utilizing the Raspberry Pi in a unique guitar effects processor, this project aims to determine the Raspberry Pi's usefulness in a real-time audio context, revealing the device's limitations as well as its strengths.

Existing Work

A handful of existing projects influenced the design of this one. Pierre Massat, who maintains a blog called Guitar Extended, uses a Raspberry Pi to process his guitar, and shares his techniques online (Massat, 2013). A Linux audio enthusiast identifying by the title Autostatic has posted extensive documentation on running audio programs on the Raspberry Pi with low latency (Audiostatic, 2014). Similarly, the project Amp Brownie documents experiments using the Raspberry Pi with the guitar effects program, guitarix (Amp Brownie, 2014).

Satellite CCRMA, a Stanford-hosted platform for creating embedded musical instruments, has also been hugely influential for this project (Berdahl & Ju 2013). Many student-created applications, such as Martin Hünniger 's “Guitar Granulator” showcased the creative usefulness of a single-board computer for audio processing (Hünniger, 2013).

Finally, Pd2live, a project by Ricky Graham, and various augmented guitar projects by Enda Bates have been very inspirational, because the sounds that result from their efforts

transcend what is possible with traditional, commercially available guitar effects pedals (Bates, 2013; Graham, 2012).

Chapter 2

Software

Choosing a Linux Distribution

Raspberry Pi users have many different choices for operating systems, the majority of which are Linux-based. The open source nature of Linux means that plenty of software is freely available, and can be modified as desired. This makes the platform a strong choice for the project.

Initially the Raspbian Wheezy distribution was chosen for the project. Links to download this distribution are provided straight from the Raspberry Pi website, and due to its wide user base, it is easy to find support. Unfortunately, an unresolved issue in the current Linux kernel for this distribution seems to cause issues with audio input for some USB audio devices (Roberts, 2013). With Raspbian Wheezy, it was impossible to use duplex audio (simultaneous input and output) at a samplerate higher than 32kHz.

For this reason, the Satellite CCRMA distribution was selected as an alternative. Satellite CCRMA was designed specifically for use with embedded audio applications. Using the Behringer UCG102 interface with Satellite CCRMA to run duplex audio at 44.1kHz has caused no issues. An additional benefit of this distribution is its ability to disable writing to the SD card, which protects memory during power cycles (Berdahl, Salasar, & Borins, 2013). This feature makes it safe to unplug the Raspberry Pi from its power source without calling any shutdown commands first.

Choosing Software for Audio Processing

Initially, digital signal processing code for this project was developed in C, using the open-source PortAudio library to handle audio input and output (“PortAudio”, n.d.). It seemed intuitive that working in this relatively low-level language would offer strong performance, which is important for a system with processing constraints.

Unfortunately, writing low-level signal processing code is very time-intensive, and it took many hours of frustrating work to develop a desired framework for linking together chains of audio effects. While simple delays and a looper were functioning after some time, the implementation was ultimately a failure.

Pure Data (Pd), a graphical programming environment, was finally selected as an alternative to writing native code (Pure Data, 2013). Programming in Pure Data involves “patching” together various unit generator objects, which create and process signals. Many common signal processing operations are included with Pure Data as objects, and tools for communicating with MIDI devices are also available. These features have made it much easier and faster to develop ChainLinkFX, and there has been no noticeable drop in performance or stability.

On the Raspberry Pi, several flavors of Pure Data can be installed. “Pd vanilla” is the most stable version, and a compiled executable for the Raspberry Pi is available from UCSD (http://msp.ucsd.edu/software.html). “Pd-extended” contains the core functionality of the vanilla version, but includes many useful external objects. Additionally, a new branch of Pure Data called “Pd-L2Ork” is now available for the Raspberry Pi from the Linux Laptop Orchestra (“L2Ork Linux Laptop Orchestra”, 2014). This version features a more accessible user interface, and it includes objects for working with the Raspberry Pi's GPIO ports.

Initially, Pd-extended was used for the project. Unfortunately, the program would sometimes fail to start when requesting a low (< 20ms) audio latency. The vanilla version of Pd does not have this problem, so it has been chosen for the final implementation.

Chapter 3

Hardware

Figure 2: Hardware used in ChainLinkFX – Raspberry Pi (right), Behringer UCG102 (bottom-left), and a Cyberpower USB hub (upper-left)

Several different pieces of hardware were used to develop ChainLinkFX. Audio input and output are made possible by a Behringer UCG102 USB sound card. The Raspberry Pi itself is the centerpiece of the project. It interfaces with the other hardware in the system, and processes audio. Software on the Raspberry Pi is updated from a separate laptop computer, which makes a secure-shell (SSH) connection to the device via an Ethernet cable. When it is booted, the Raspberry Pi runs a start-up script that runs the custom software. Connecting to a laptop is not necessary for the system to run.

A Cyberpower powered USB hub connects the Raspberry Pi to both the UCG102 and the QuNeo MIDI Controller. This ensures that the QuNeo and UCG102 receive stable power. Additionally, the USB hub provides power to the Raspberry Pi itself. Connecting the power supply of the USB hub to a wall outlet will provide all necessary power for the system.

For additional control, a momentary push button is connected to the GPIO pins on the Raspberry Pi, with one end attached to pin GPIO4, and the other attached to a ground pin. The University of Cambridge provides a helpful page explaining how to set up this circuit (Andrade, Jones, Lee, & Bates, n.d.).

Quarter inch TRS cables connect the guitar to the UCG102, and the UCG102 to an amplifier. The various devices are kept in a plastic box. A hole drilled in the side of the box allows cables to pass through, and a hole in the top creates a space to house the button.

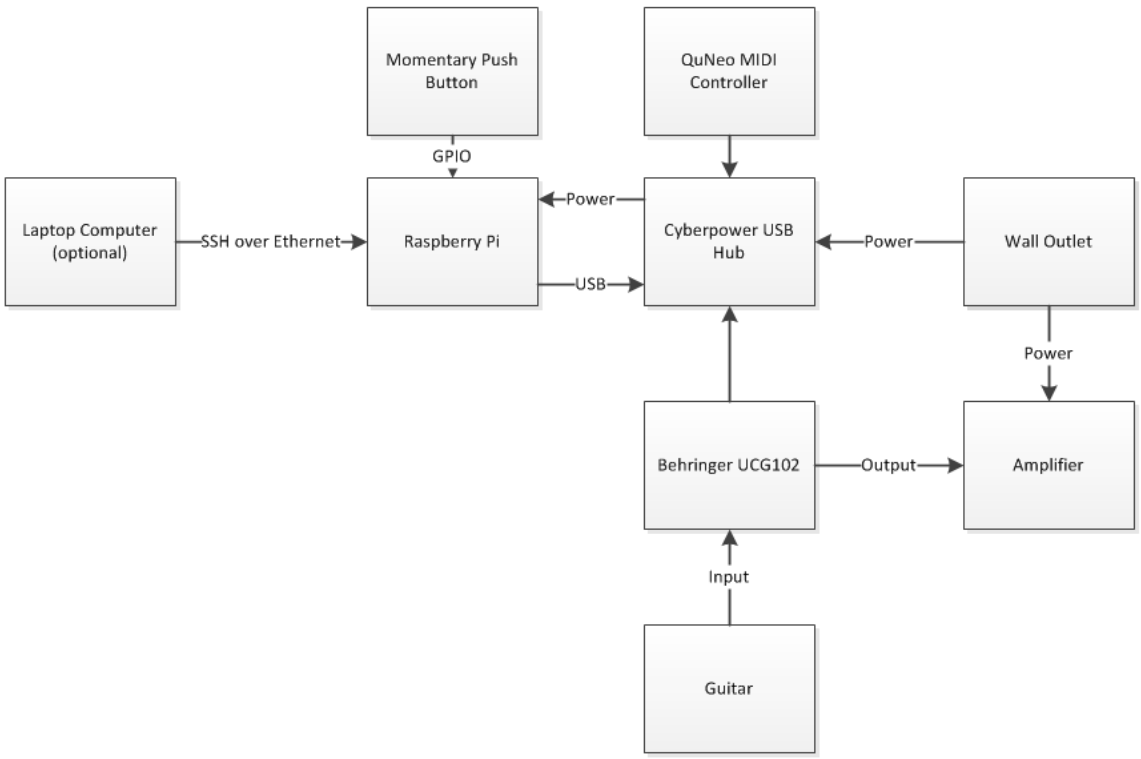


Figure 3: Hardware Diagram of the various components used in ChainLinkFX

Chapter 4

Implementation

System Design

Guitarists often run their signal through a chain of several effects pedals. The order of the effects can make a difference in the sound, and most pedals have a “bypass” switch to halt their processing. By switching between two chains of effects, a guitarist can instantly change from tone to another.

ChainLinkFX was designed with this flexibility in mind. Two chains of effects are run in parallel. Each chain can hold up to eight serial effects, and volume for each chain can be independently controlled. One benefit of digital effects processing is that no wires are required. Effects can be reordered instantaneously, without fiddling with cables, and multiple instances of the same effect can be used.

Control

Effects and their parameters are adjusted using a momentary push-button and a MIDI controller. To detect button presses, a custom external for Pure Data was written in C. This external defines a Pure Data object that checks every millisecond for changes in the state of the button, and outputs a trigger message every time it is pressed down. Gordon Henderson's wiringPi library was used to poll the state of the Raspberry Pi's GPIO pins (Henderson, 2013).

A custom preset was created for the QuNeo MIDI controller for use with the project. Local LED control is disabled for many of the controls, allowing the LEDs to be set by Pure Data. The sixteen pads on the controller are set to “grid mode,” which sets the four individual

corners of each pad to unique MIDI notes. This way, the grid of pads is split into eight horizontal rows of eight controls. Two of these rows are used to represent the two effects chains, with each corner corresponding to an individual link on the chain. The leftmost corner represents the first link in the chain, and the rightmost represents the last link.

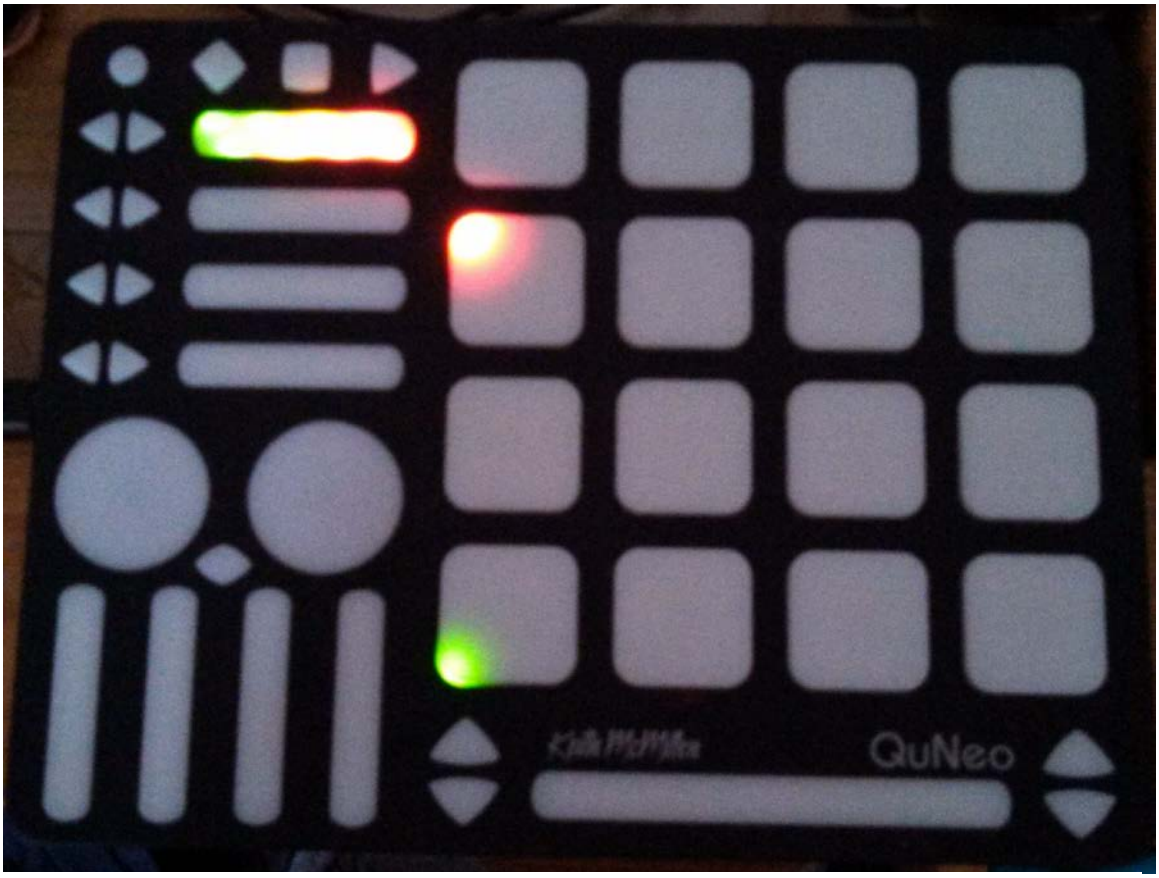


Figure 4: The QuNeo interface. The red light in the upper-left corner of the upper pad indicates that the first effect of the first chain is selected. The green light in the lower-left corner of the lower pad indicates that the effect is set to bypass. The illuminated slider indicates the current volume of the first chain, which is at full percentage.

When a corner from one of these two rows is pressed, it is lit up with a red light, informing the user that the corresponding link in the effects chain has been selected. One of eight corners on the bottom row will also be illuminated with a green light, representing the effect that is currently selected for that link. Pressing another corner of the bottom row will select a different effect.

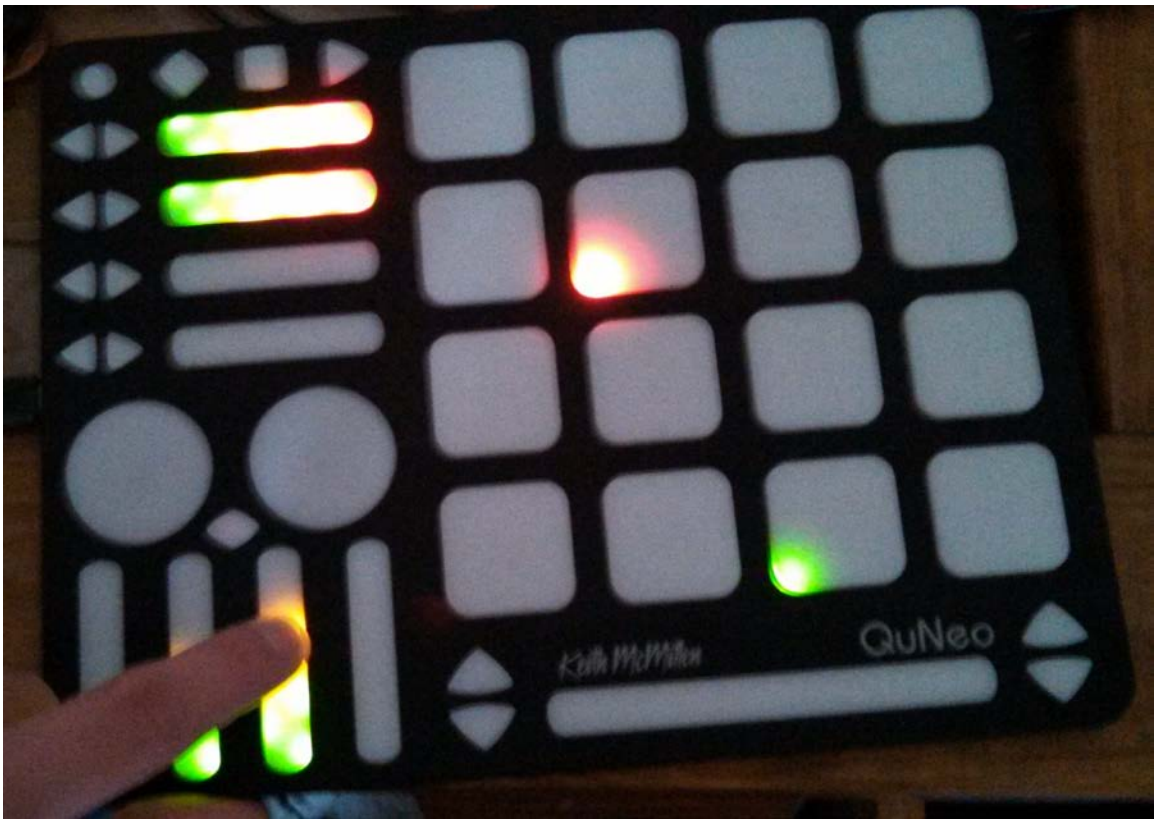


Figure 5: The red light on the upper pad indicates that the third effect on the second chain is currently selected. It is set to a flanger, which is indicated by the green light on the lower pad. The two horizontal sliders represent the current volume of each effects chain. The vertical sliders correspond to parameters of the currently selected effect. Currently, the feedback level of the flanger is being adjusted.

Chapter 5

Effects

Looper

The looper allows a passage to be recorded and repeated endlessly. One button press with a foot begins the recording. A second press stops the recording and begins playback of the loop. Up to three minutes can be recorded per loop.

After a loop has been recorded on one chain, the user may desire to play alongside the recorded audio, using a different tone. When this effect is selected, the leftmost vertical slider on the QuNeo sets the gain on the input to the looper. Turning down the input gain will stop the input signal from passing through the rest of the chain, while allowing the looped audio to continue playback. This way, users have the flexibility to play on top of a loop using a completely different tone, using the same exact tone, or using some combination of multiple tones.

Only one loop is available per chain, but it may be placed anywhere in the series of eight effects on that chain. This leads to interesting possibilities. If the looper is placed first in the series of effects, later effects in the series can be used to adjust the tone of the recorded audio. If the looper is placed last in the series of effects, it will record the output of the previous effects. A looper can even be repositioned in the chain as it is playing – it could potentially record the output of a series of seven effects, and then be placed back at the beginning of the chain to pass through another seven effects. Doing this with traditional guitar effects pedals is impractical, as it would require rerouting a bunch of cables mid-performance. With ChainLinkFX, only several button presses are necessary.

Delay

This effect simply takes an input signal and adds a delayed copy to it. The delayed signal is fed back into itself, so that multiple echoes are heard. The leftmost vertical slider on the QuNeo sets gain to the delay – at the lowest level, nothing is fed into the delay and the input signal is unaffected. At the highest level, many echoes will be heard before the delay decays to inaudibility. Delay time is set by tapping the push button with the desired tempo. The maximum delay time is one second.

Distortion

This effect distorts the input signal by passing it through a transfer function. The transfer function used is a cubic nonlinearity that smoothly clips the signal (Smith, 2010a). The leftmost vertical slider on the QuNeo sets the input gain to the transfer function, determining the level of distortion. The second slider controls the output gain, which is set to a low value by default to approximately match the loudness of the input signal.

An envelope follower is used to track the level of the input signal. If it is below a certain threshold, the input is silenced. Without this gate, noise would pass through the distortion when the user is not playing anything. At a low input gain, the distortion adds a small “crunch” to the guitar tone. More extreme distortion can be achieved by chaining two instances of this effect together, resulting in a sound similar to a square wave.

Flanger

The flanger uses a variable delay line, modulating between around 1 and 5 milliseconds. This creates a shifting comb-filter that adds some movement to the spectrum of the input signal.

The leftmost vertical slider on the QuNeo is used to control the modulation rate, between 0 and 1 hertz, the second slider controls the depth of the modulation, and the third slider sets the level of feedback to the delay line. This effect is most noticeable when it is placed after a distortion, which adds harmonics to the input signal and creates a wider spectrum to be affected by the flanger.

Granular

The next effect in ChainLinkFX is a simple form of granular synthesis. The input signal is written to a circular buffer. “Grains” triggered at a set rate quickly fade in and out the playback of this buffer, using a Hanning window as an amplitude envelope. Buffer playback randomly switches between normal, half, and double speed to create shifting octaves. The leftmost vertical slider on the QuNeo controls interonset time of the grains, the next slider sets the probability of a grain being triggered, the third slider sets the duration of the grain, and the final slider controls wet/dry mix.

This implementation of granular synthesis is fairly limited, in order to minimize performance issues. Grain scheduling is synchronous, only occurring at a set interval, but probability control adds a stochastic element to the effect. The amplitude window cannot be adjusted, and only two grains are used, so there is not much room for grains to overlap. The effect barely touches the surface of possible applications of granular synthesis. In future versions of the system, a more versatile approach to granular synthesis could be adapted from Ross Bencina’s implementation, which offers different sequencing strategies and more controllable parameters. (Bencina, 2001).

Despite its simplicity, the effect can be put to creative use. It can be applied subtly and passed through delays or reverberation to add interest to a solo tone. Setting the grain duration

and interonset time to very small intervals can create strange bubble-like sounds. With a probability of one-hundred percent, the grain playback occurs with a consistent rhythm. Lowering the probability generates a more unpredictable output.

Reverb

The reverb used in the project is an implementation of the “Freeverb” algorithm, which is a Schroeder reverberator that uses allpass and comb filters (Smith, 2010b). The patch for this reverb, which is compatible with pd vanilla, was adapted from a patch provided by Katja Vetter in the Pure Data forums (Vetter, 2013).

The leftmost vertical slider on the QuNeo sets the room-size parameter, the second slider controls damping, the third slider controls the gain of the dry output signal, and the final slider sets the gain of the wet signal. Additionally, the push button can be pressed to toggle input to the reverb on and off. This is useful for selectively sustaining notes and chords.

EQ

The final effect is a simple three-band EQ, which uses biquad filters to modify the spectrum of the output signal. The patch for this EQ was posted by the Pure Data forum member, “hardoff” (Hardoff, 2008). The leftmost vertical slider on the QuNeo controls the level of bass output, the second slider sets the level of the mids, and the third slider sets the treble. This effect is especially useful for coloring the output of a distortion or cutting out excessive bass from the granular and reverb effects.

Chapter 6

System Performance

With reasonable use, audio performance is stable, with no noticeable glitches. However, using too many effects simultaneously puts serious strain on the Raspberry Pi. Running four or five of the more computationally intensive tasks, like the Flanger, Reverb, and Granular effects, may result in audio dropouts that completely ruin the signal. Overclocking the Raspberry Pi seems to help raise the number of effects that can be run simultaneously, but it comes with added risk of corrupting the SD card. It is possible that optimizations could be made to the various effects, perhaps by rewriting them as C externals.

An earlier implementation of ChainLinkFX used four parallel chains of effects, rather than the two it uses currently. As more effects were implemented, this required more increased memory and processing power. The decision to reduce the system to two chains greatly improved stability. Also, it was hard to imagine scenarios where four simultaneous chains would be necessary. Concentrating on two unique chains of effects is difficult enough for the performer, and ultimately this limitation may actually be an improvement. In future versions of ChainLinkFX, the number of serial effects may be reduced from eight to four, as it is rare to actually desire eight effects in a single chain.

The latency setting in Pure Data is set to ten milliseconds. Overall system latency may be slightly longer than that, due to delays inherent in the input and output of the sound card. The delay is too short to be perceived as an echo, but it is somewhat noticeable, especially when playing fast rhythmic passages. Some guitarists may feel absolutely comfortable with the amount of latency, while others may dislike performing with it. Increasing the amount of latency could

lead to improvements in system performance, but it would make it difficult for the user to play with precise timing. Requesting a smaller latency will result in more frequent audio glitches.

Comparing the sound of a guitar fed straight into an amplifier to that of a guitar first passed through ChainLinkFX, there does seem to be some difference in tone. The output of ChainLinkFX has additional noise, but it is only audible when the guitar is not playing. Some of these inconsistencies in tone may be explained by impedance mismatch; the output signal of an electric guitar has a different impedance than that of the UCG102. A transformer placed between the UCG102 and the amplifier could potentially improve the final sound.

Chapter 7

Conclusion

Despite its somewhat limited processing capabilities, the Raspberry Pi was used successfully to implement a unique guitar effects processor. ChainLinkFX incorporates some effects that are already commonly used by guitarists. In addition, the system is useful in some ways that traditional guitar effects pedals are not. Multiple chains of effects can be used simultaneously, and effects can be reordered and tweaked with simple button presses. The ability to adjust the order of effects creates a huge array of possibilities in terms of sounds. If new features are desired, the system can always be reconfigured to add new effects or to optimize performance.

In other ways, ChainLinkFX is lacking. It adds some noise and latency to the guitar signal. Pushing the system too far leads to audio glitches, but it is hard to predict when they will occur. Software optimization could help reduce these glitches, but ultimately there are limits to what the hardware can do.

The capability of the Raspberry Pi device gives reason to believe that similar single-board computers will perform as well or better. Alternatives currently include the UDOO board, the Beaglebone Black, and the CubieTruck, which all have more processing power than the Raspberry Pi. As the popularity of these computers increases and the experiences of the creators who use them are shared, new creative musical projects will likely emerge.

For someone interested in the Raspberry Pi as an alternative to a traditional effects pedal, the customization that the computer allows is a double-edged sword. It takes time and effort to set up the hardware and software, and it may be impossible to exactly emulate the functionality of an existing pedal. However, the freedom that the Raspberry Pi establishes allows the creator to make something truly unique. The successful implementation of ChainLinkFX merely hints at the possibilities of the hardware. There are few things as enticing to musician as creating new sounds, and the computer has once again demonstrated its value in this function.

Chapter 8

Appendices

Appendix A

Buttonlistener external

The buttonlistener external listens for changes in the state of the button attached to GPIO pin 17. When Pd is started, buttonlistener_setup() is called, initializing wiringPi and defining the buttonlistener class. One buttonlistener object is used in the Pd patch, and when it is initially

```c
#include "m_pd.h"
#include <wiringPi.h>

t_class *buttonlistener_class;

typedef struct _buttonlistener
{
    t_object x_obj;
    t_clock *x_clock;
    int defaultButtonState; //some buttons close switch when pressed, others open it, might as well make the code neutral.
    int buttonState;
    int pinNum;
} t_buttonlistener;

void buttonlistener_tick(t_buttonlistener *x)
{
    int prevState = x->buttonState;
    x->buttonState = digitalRead(x->pinNum);
    if(prevState != x->buttonState){
        if(x->buttonState != x->defaultButtonState){ //bang if button has just been pressed
            outlet_bang(x->x_obj.ob_outlet);
        }
    }
    clock_delay(x->x_clock, 1); //polling every millisecond
}

void *buttonlistener_new()
{
    t_buttonlistener *x = (t_buttonlistener *)pd_new(buttonlistener_class);
    x->x_clock = clock_new(x, (t_method)buttonlistener_tick);
    outlet_new(&x->x_obj, gensym("bang"));
    //call the tick function once to start it
    x->pinNum = 0; //0 is GPIO 17 in wiringPi
    pinMode(x->pinNum, INPUT);
    x->buttonState = digitalRead(x->pinNum);
    x->defaultButtonState = x->buttonState;
    buttonlistener_tick(x);
    return (x);
}

void buttonlistener_free(t_buttonlistener *x)
{
    clock_free(x->x_clock);
}

void buttonlistener_setup()
{
    wiringPiSetup();
    buttonlistener_class = class_new(gensym("buttonlistener"),(t_newmethod)buttonlistener_new,
        (t_method)buttonlistener_free, sizeof(t_buttonlistener), CLASS_NOINLET, 0);
}
```

Figure 6: buttonlistener.c, a Pure Data external that checks for button presses.

loaded, buttonlistener_new() is executed. This initializes the buttonlistener object, tells wiringPi to listen to GPIO 17 (numbered as pin 0 in wiringPi), and finally executes buttonlistener_tick(). This function checks the button state, outputting a bang to the object's outlet if it has changed since the last tick. The buttonlistener_tick() function is called every millisecond until the object is destroyed. This code borrows heavily from the source code of Pd's metro object, which is defined in the source file x_time.c.

One millisecond is a short enough interval of time that the user will not notice a delay between pressing the button and hearing a result. Using a smaller interval of time would take up more valuable CPU time, and could also expose moments when the state of the button is in flux, wavering between on and off.

Appendix B

Pure Data Patches

This section documents the various Pure Data patches that are used in ChainLinkFX, providing screenshots and a general overview of each. While these patches should run on any copy of Pd Vanilla, their functionality is largely dependent on the connected hardware. Any interested reader may download these patches for personal use and modification alongside this document. The buttonlistener external code and a custom preset for the QuNeo controller are also available. Additionally, the most up-to-date version of all code will always be available at the author's Github account, https://github.com/YottaSecond.

mainDevelopment.pd

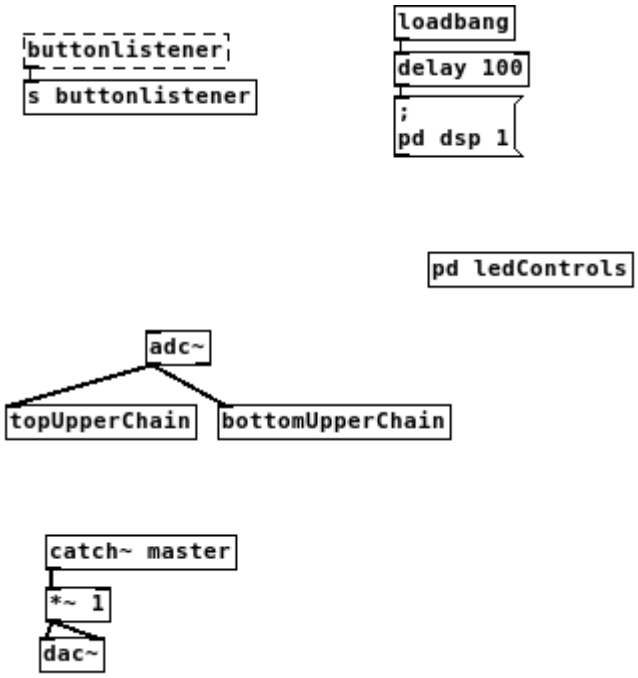


Figure 7: mainDevelopment.pd

This patch is the root of ChainLinkFX. A [loadbang] turns dsp on after a delay of 100 milliseconds. This allows the patch to be started from the command line without using Pd's GUI. The guitar input signal arrives at the left inlet of [adc~], and passes into [topUpperChain] and [bottomUpperChain]. The [buttonlistener] sends its output to any receiver objects listening for messages subscribed to the title "buttonlistener." This object appears in dashed lines because it has not been compiled on the laptop from which this screenshot was taken. Output of the two chains is thrown to the "master" bus, which is output to both channels of the [dac~] for monophonic output. The [*~ 1] object is in place so that a master volume control could be implemented in the future, if desired.

pd ledControls(subpatch)

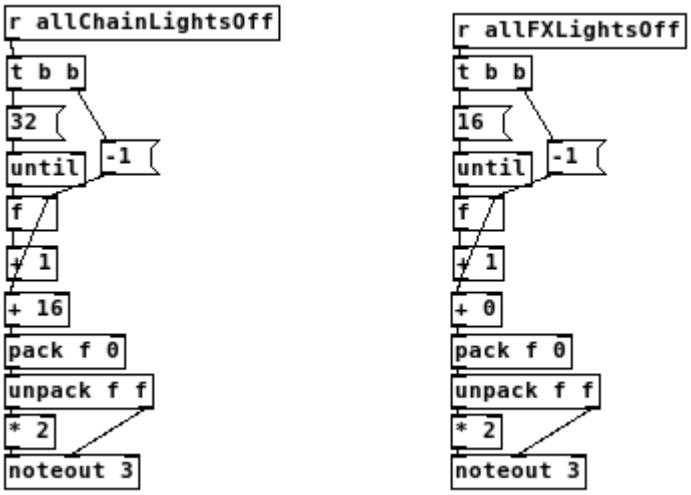


Figure 8: LED controls in the main patch

This subpatch controls the LEDs of the grid buttons on the QuNeo. When [r allFXLightsOff] receives a message, it turns off all lights in the lower row of buttons. The rest of the rows are turned off when [r allChainLightsOff] receives a message.

topUpperChain.pd

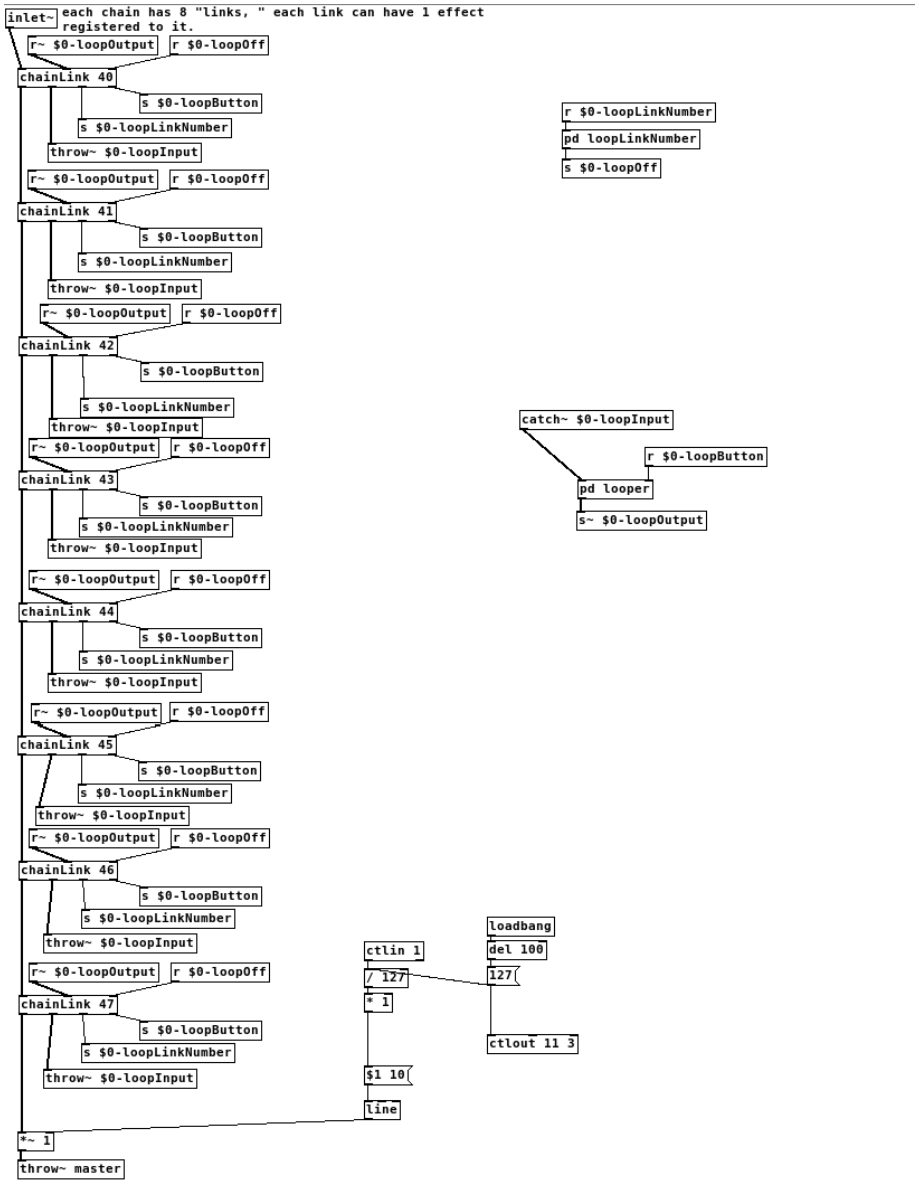


Figure 9: topUpperChain.pd

This patch contains one chain of eight effects. Each [chainLink] is an effect. The numeric argument of [chainLink] corresponds to MIDI note the object should listen for. The [line] connected to the [*~ 1] allows volume to be controlled via MIDI input.

All of the [s] and [r] objects attached to the [chainLink] objects are related to the chain's loop. Because there is only one loop per chain, it is handled differently than the other effects in the system, which are held inside the [chainLink] objects.

The bottomUpperChain.pd patch is almost identical to this patch, only differing in MIDI note numbers and initial volume, so it will not be pictured in this appendix.

pd loopLinkNumber(subpatch)

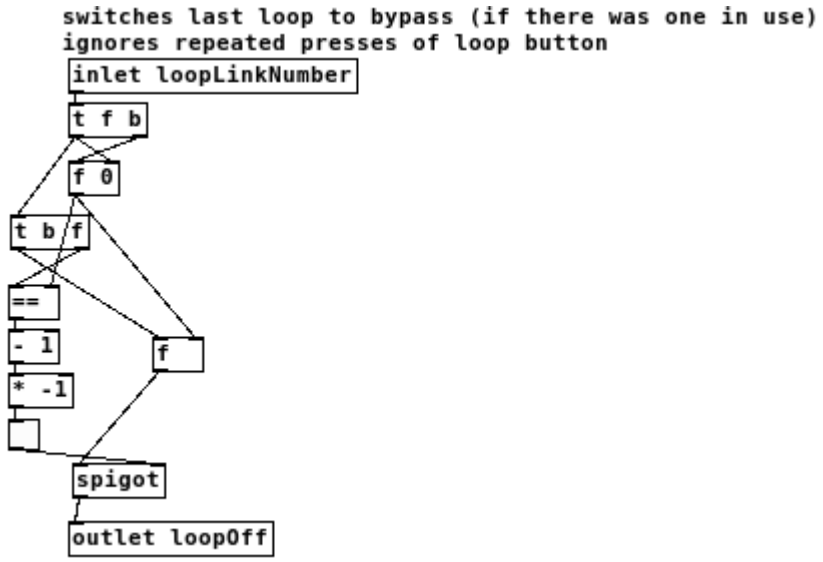


Figure 10: pd loopLinkNumber

The subpatch, "pd loopLinkNumber," accounts for a situation where the user switches the looper from one link in the chain to another. If the user repeatedly presses the looper effect button, nothing will happen.

pd looper(subpatch)

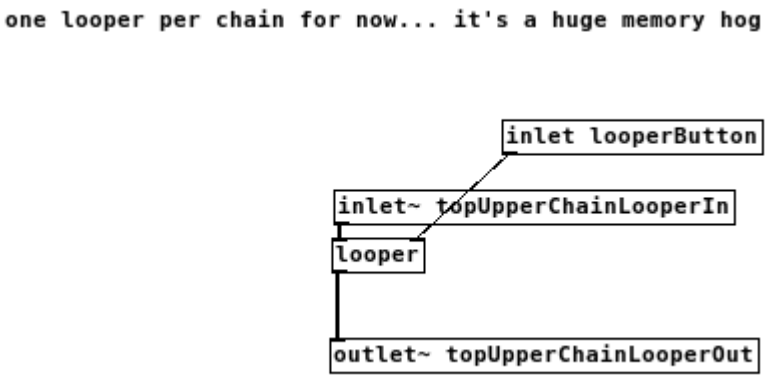


Figure 11: pd looper

This subpatch merely receives audio and button input from a [chainLink] that has been set to a looper effect, and sends the output of the looper back into the [chainLink].

chainLink.pd

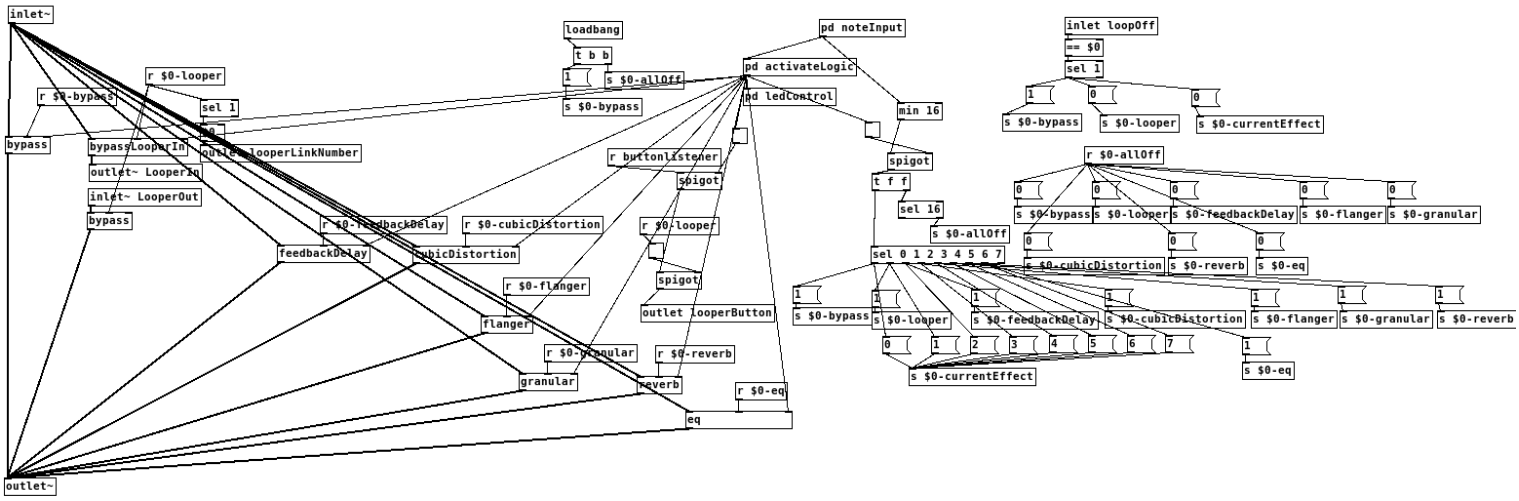


Figure 12: chainLink.pd

By far the ugliest patch in the system, chainLink.pd is perhaps an example of graphical programming done poorly. In future releases, the patch should be broken down into more manageable subpatches, to reduce the visual clutter.

Each [chainLink] object contains seven effects (everything except the looper), and one bypass object. The input signal is sent into all effects, but only one can be active at a time. This portion of the code is handled on the left side of the patch. The right side of the patch deals with MIDI messages. If the chainLink is currently selected, pressing the bottom row of MIDI controls will switch the active effect. Otherwise, these Note On messages will be ignored.

pd noteInput(subpatch)

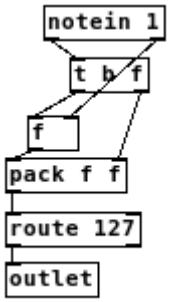


Figure 13: pd noteInput

This subpatch simply listens for incoming MIDI Note messages on Channel 1, and outputs them if they have a velocity of 127 (QuNeo button presses are currently set to output only 127).

pd activateLogic(subpatch)

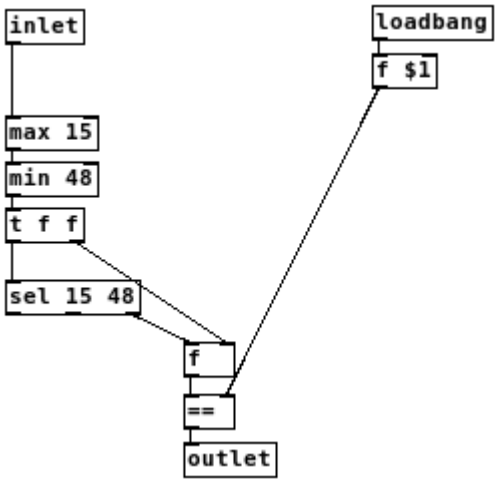
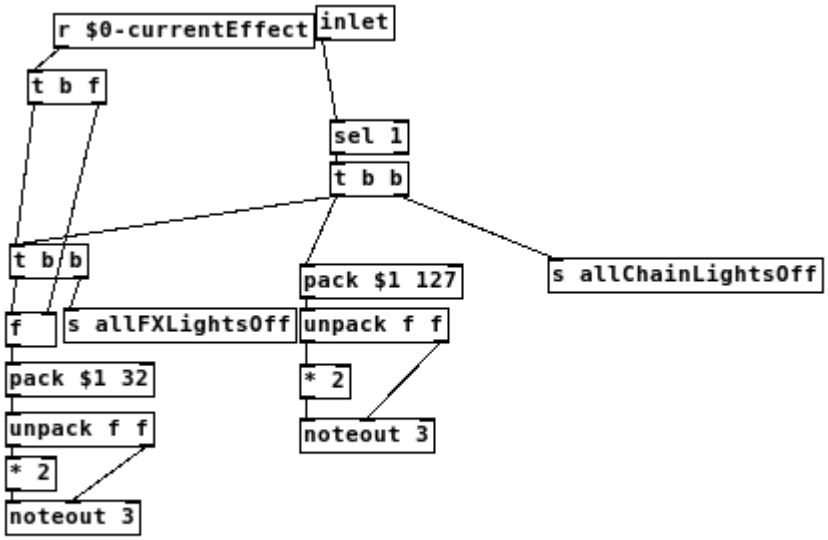


Figure 14: pd activateLogic

The [loadbang] in this subpatch initializes the [==] with the number outputted by [f $1], which is the argument to the [chainLink] object. If a MIDI Note On has the same note number as this argument, the [chainLink] will become "active," so that its effect can be changed or its

parameters can be adjusted. If one of the buttons on the chain was pressed, but it is not the same as this argument, the [chainLink] will deactivate.

pd ledControl(subpatch)



This subpatch responds to the output of [pd activateLogic]. If the [chainLink] is activated, it will first turn off all other grid lights, and then turn on the lights corresponding to the [chainLink] and its currently selected effect.

bypass.pd

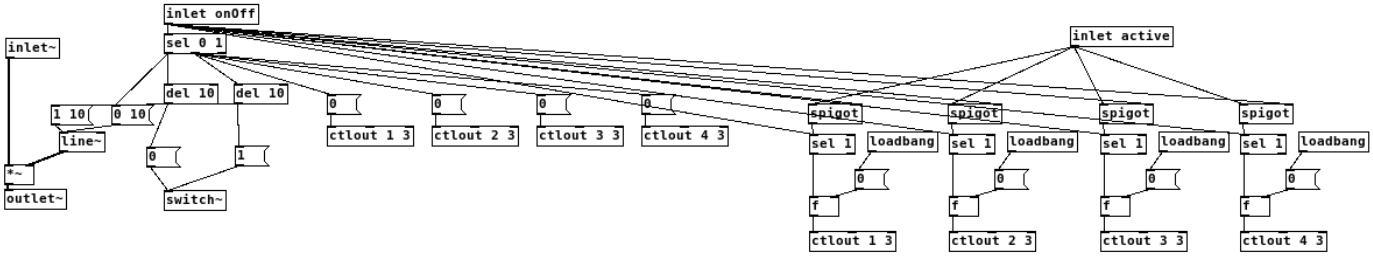


Figure 15: bypass.pd

This patch is the default of the eight possible states of [chainLink]. It simply takes the input signal and sends it to the output. When activated, the [switch~] object is turned on, enabling audio processing for the patch. Ten milliseconds later, audio for the patch is faded in.

MIDI note messages with zero velocity are sent to the vertical sliders of the QuNeo every time this patch is activated, indicating that there are no parameters to adjust.

looper.pd

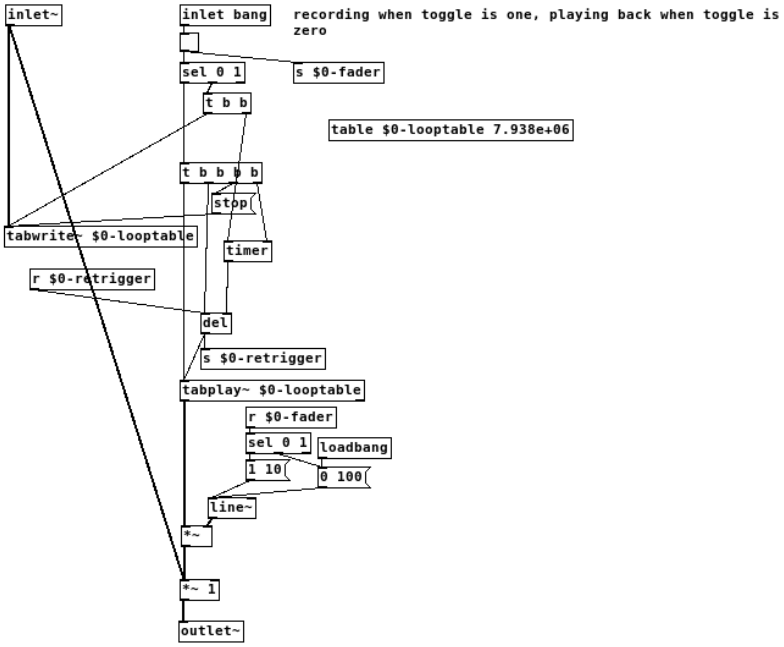


Figure 16: looper.pd

The looper patch differs from the other effects, because its activation logic is handled outside the patch (see topUpperChain.pd). The patch contains a table that holds three minutes worth of samples. When it receives a bang, it begins recording audio into the table, keeping track of how much time has elapsed with a [timer]. Another bang causes the table to be played back. Table playback is retriggered at the rate recorded by the [timer]. If the user attempts to record a four minute loop, he or she will hear three minutes of playback, followed by one minute of silence before the loop repeats.

feedbackDelay.pd

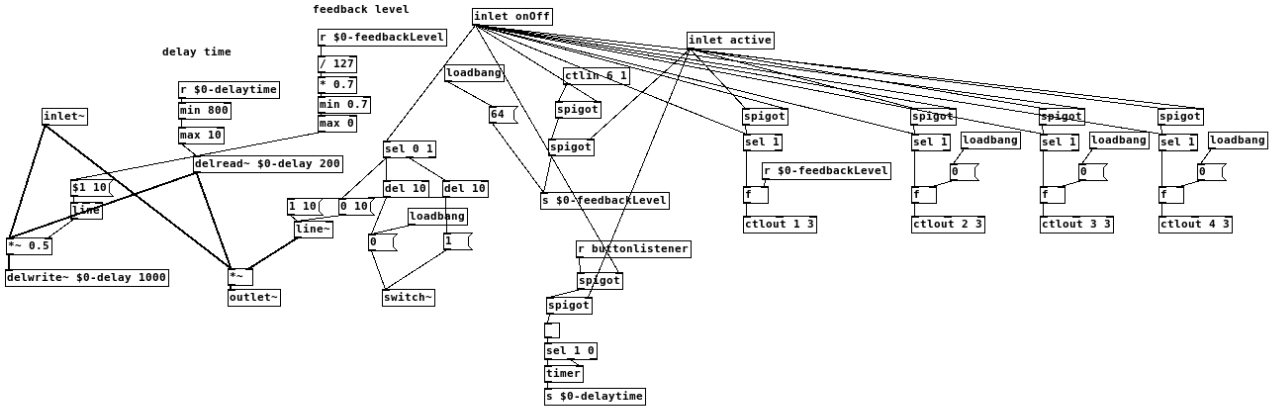


Figure 17:feedbackDelay.pd

Most important to this patch are the [delwrite~] and [delread~] objects, which record input to a buffer and read it back after a specified interval of time. The output of the delay reader is fed back to the writer at an amplitude that can be adjusted between 0 and 0.7 times the input signal.

cubicDistortion.pd

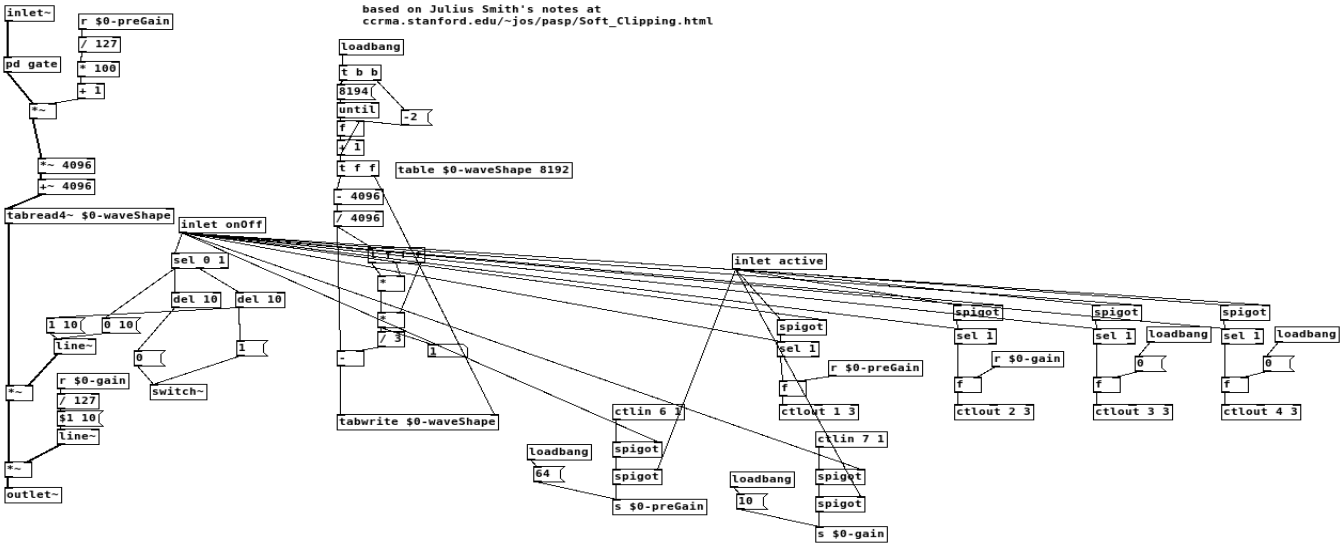


Figure 18: cubicDistortion.pd

In the center of the patch, a control loop writes a cubic nonlinearity function to a table containing 4096 points. This is used as a transfer function that distorts the input signal. The gain-before-distortion can be adjusted from one to one hundred times the amplitude of the original input, effectively determining the level of distortion.

pd gate(subpatch)

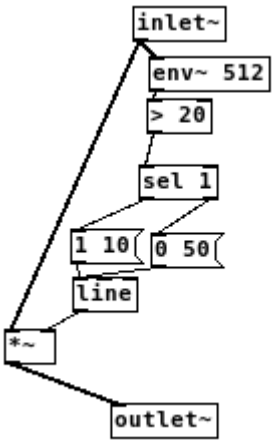


Figure 19: pd gate

This subpatch uses an [env~], which follows the level of the input signal. When the input signal drops below a certain threshold (determined via trial and error), the input is faded out to zero amplitude. This eliminates background noise when the guitarist is not playing.

flanger.pd

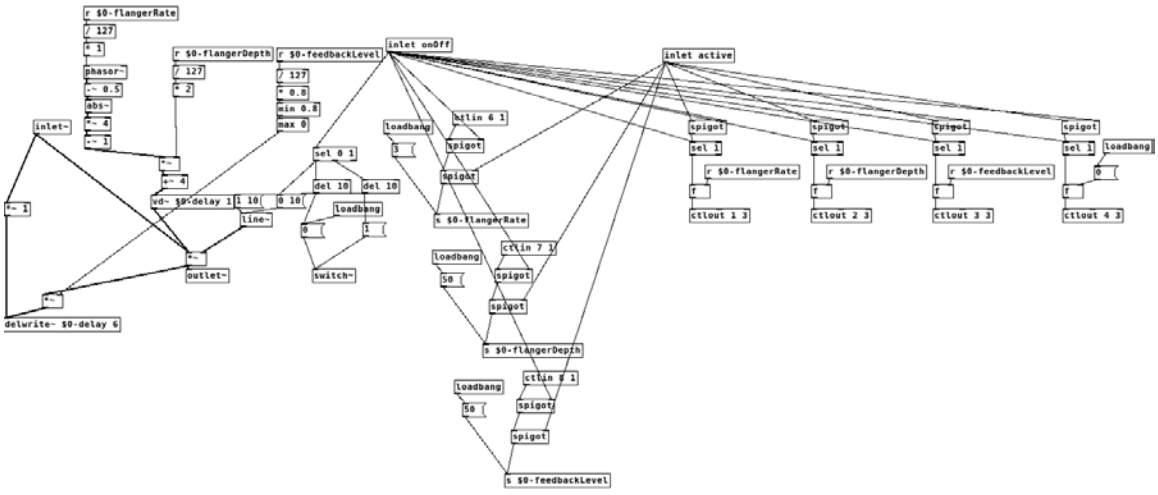


Figure 20: flanger.pd

The flanger patch emulates a tape flanging effect, using a variable delay line that is modulated by a triangle wave oscillator. The depth of the delay modulation can be adjusted between zero and four milliseconds of difference. The modulation rate can be set between zero and two hertz. Feedback to the delay line can also be adjusted.

granular.pd

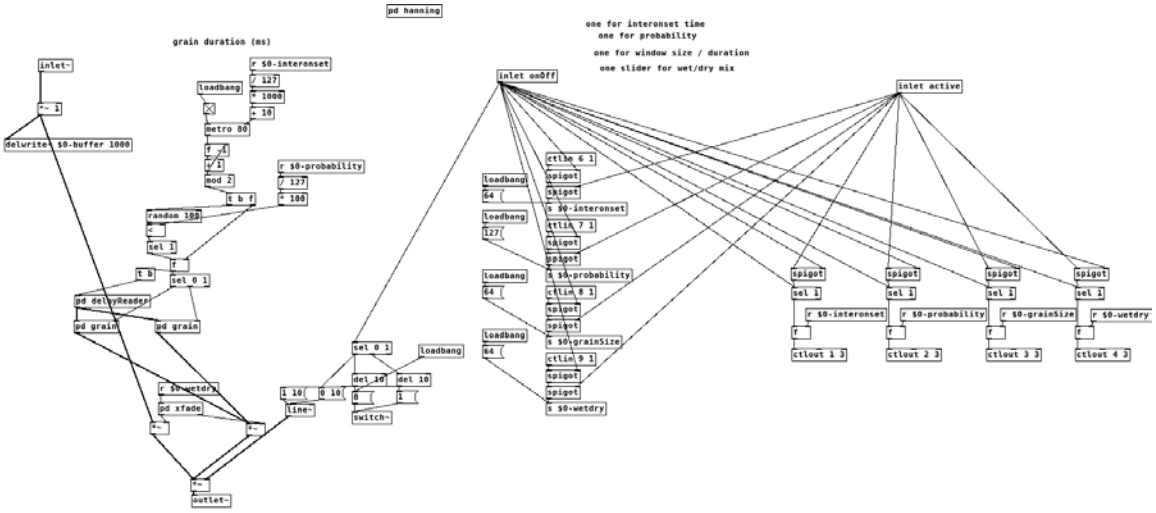


Figure 21: granular.pd

The granular patch is a bit more complicated than most of the other effects. The signal is written to a buffer with the [delwrite~] object. The delayReader subpatch reads back this buffer at different pitches, and outputs its signal to the two grain subpatches. A [metro] triggers pitch changes in the delayReader subpatch, and alternatively triggers each grain (this allows them to overlap). The tempo of the [metro] can be adjusted, and so can the probability of its bang messages being sent onward. The duration of each grain is set as a fraction of the tempo of the [metro], so grains can be longer when the tempo is slower.

pd delayReader(subpatch)

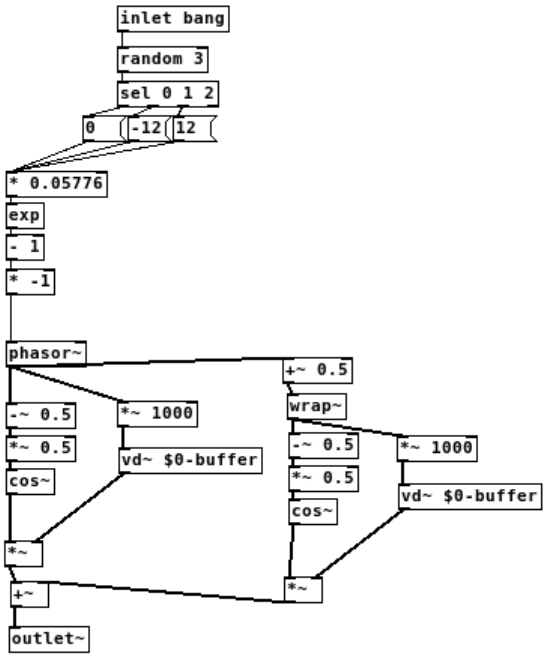


Figure 22: pd delayReader

This subpatch reads through the buffer using two variable delays. These delay lines fade in at the beginning of the circular buffer, and out when they reach the end, to avoid stuttering effects. The two delay lines are positioned out of phase with one another, so that one is always audible. This algorithm borrows heavily from G08.pitchshift.pd, an example patch that is included with Pd (Puckette, 2003).

pd grain(subpatch)

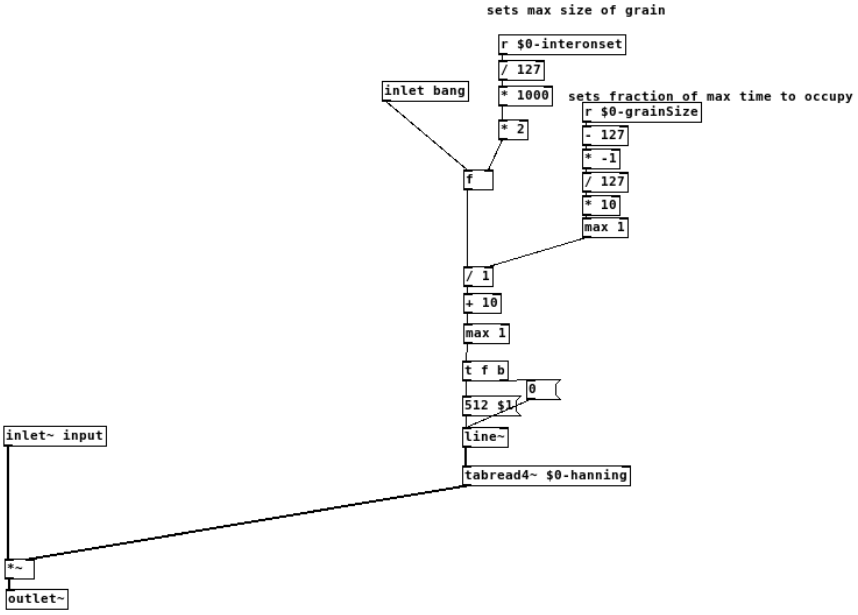


Figure 23: pd grain

This subpatch implements one grain, fading in the output of [pd delayReader] when triggered. Its amplitude envelope is a Hanning window. The grain size is calculated here as a fraction of the interonset time. Two of these subpatches are used, to enable overlapping grains.

pd hann(subpatch)

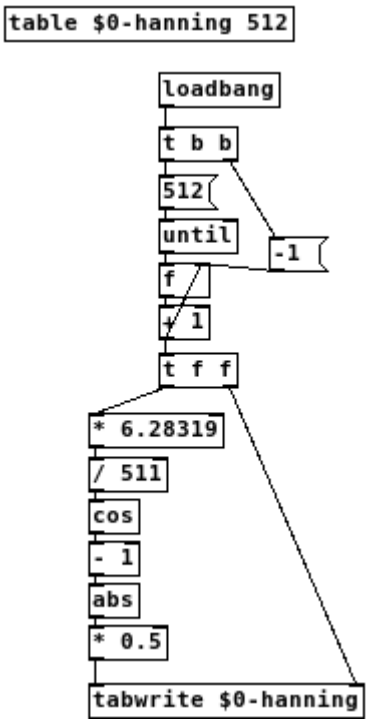


Figure 24: pd hann

This subpatch draws the Hanning window that functions as the amplitude envelope of each grain. It is a smooth curve drawn using a portion of a cosine wave.

pd xfade

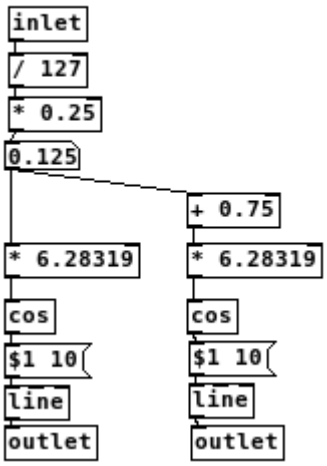


Figure 25: pd xfade

This subpatch is used to crossfade the wet/dry mix of the effect. It attempts to achieve equal output volume no matter what the ratio of wet to dry levels is.

reverb.pd

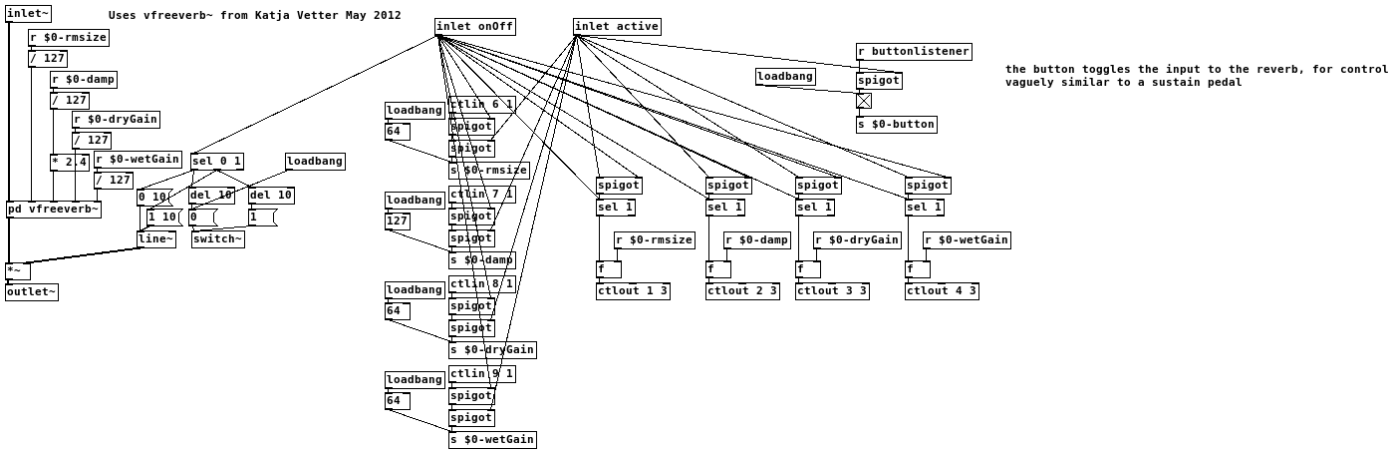


Figure 26: reverb.pd

This patch is an implementation of freeverb, an algorithm that was adapted to Pd Vanilla by Katja Vetter. The reverberation algorithm is beyond the scope of this document, but it involves

parallel comb filters fed into a series of allpass filters. Conveniently, the original patch had four controllable parameters, room size, damping, dry gain, and wet gain. These mapped nicely to the four vertical QuNeo sliders. When the patch is active, pressing the push button toggles input to the reverb, which is set inside [pd vfreeverb~] (not pictured).

eq.pd

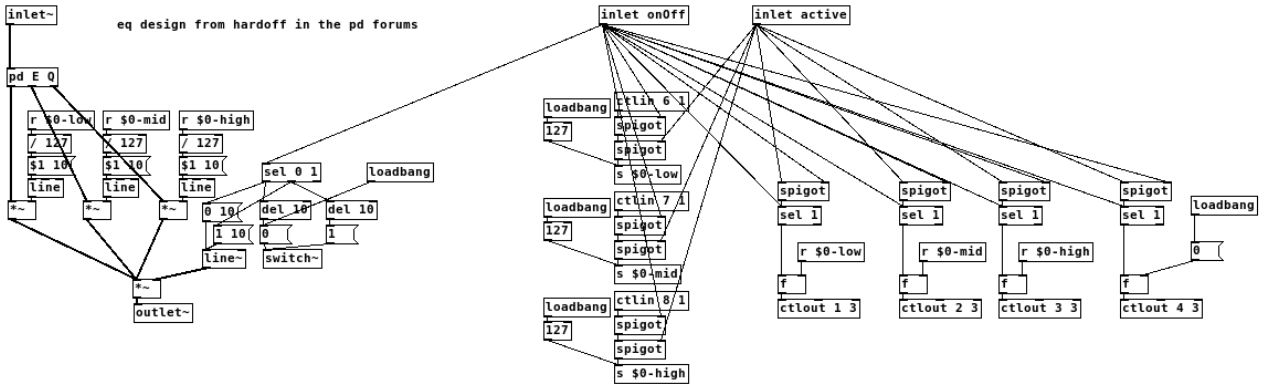


Figure 27: eq.pd

This equalizer was provided by the Pd forum member, “hardoff,” who posted the patch in response to a question about efficient EQ implementation. The guts of the [pd E Q] subpatch are beyond the scope of this text, but they contain biquad filters with specially tuned coefficients.

BIBLIOGRAPHY

Amp Brownie (2014). Amp Brownie | Raspberry Pi + Guitar Effects. Retrieved March 27, 2014 from http://ampbrownie.com/

Andrade, O., Jones, E., Lee, A. & Bates, D. Buttons and Switches. *Physical computing with Raspberry Pi.* Retrieved April 3, 2014 from http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/robot/buttons_and_switches/

Audiostatic. (2014, March 15). Rsapberry Pi and realtime, low-latency audio [Linux-Sound]. Retrieved April 2, 2014 from http://wiki.linuxaudio.org/wiki/raspberrypi

Bates, E. (2013). Augmented Guitar. Retrieved from http://www.endabates.net/augmentedguitar.html

Bencina, R. (2001, August 31). Implementing Real-Time Granular Synthesis. Retrieved from http://www.g-audio-unity.com/g-audio-content/PDFs/Implementing_Granular_Synthesis.pdf

Berdahl, E. & Ju, W. (2014). *Satellite CCRMA.* Retrieved Mar 18, 2014 from https://ccrma.stanford.edu/~eberdahl/Satellite/

Berdahl, E., Salazar, S. & Borins, M. (2013, May). Embedded Networking and Hardware-Accelerated Graphics with Satellite CCRMA. Paper presented at the International Conference on New Interfaces for Musical Expression. Retrieved from https://ccrma.stanford.edu/~eberdahl/Papers/NIME2013SatelliteCCRMA.pdf

Graham, R. (2012, March 1). Pd2Live – A Digital Music Performance System. Retrieved from http://rickygraham.com/2012/03/01/pd2live-a-digital-music-performance-system/

Hardoff. (2008, April 13). Best EQ for save CPU load [Msg 8]. *Pure Data Forum~*. Retrieved from http://puredata.hurleur.com/sujet-1687-best-save-cpu-load

Henderson, Gordon (2013, May 14). WiringPi. Retrieved from https://projects.drogon.net/raspberry-pi/wiringpi/

Hünniger, M. (2013). *Guitar-Granulator Demo* [Video file]. Retrieved from http://vimeo.com/48738498

L2Ork Linux Laptop Orchestra. (2014). Retrieved April 3, 2014 from http://l2ork.music.vt.edu/main/

PortAudio – an Open-Source Cross-Platform Audio API. Retrieved April 3, 2014 from http://www.portaudio.com/

Puckette, M. (2003, December 8). Pitch Shifter. *Theory and Techniques of Electronic Music* [Online book]. Retrieved from http://msp.ucsd.edu/techniques/v0.04/book-html/node115.html

Pure Data (2013, April 28). Pure Data – PD Community Site. Retrieved April 3, 2014 from http://puredata.info/

Raspberry Pi Foundation. (2014, April 2). Raspberry Pi. Retrieved April 2, 2014 from http://www.raspberrypi.org/

Roberts, C. (2013, August 23). USB Audio setting sample rate broken [Msg 1]. Message posted to https://github.com/raspberrypi/firmware/issues/197

Keith McMillen Instruments. (2012). *QuNeo 3D Multi-touch MIDI Pad Controller Tour Page*. Retrieved March 27, 2014 from http://www.keithmcmillen.com/QuNeo/tour

Maker Media. (2013). Maker Media | Leading the Maker Movement. Retrieved March 25, 2014 from http://makermedia.com/

Massat, P. (2013) Guitar Extended | A (possible) future of guitar. Retrieved March 25, 2014 from http://guitarextended.wordpress.com/

Smith, J. O. (2010a). Soft Clipping. *Physical audio signal processing: for virtual musical instruments and audio effect* [Online book]. Retrieved from https://ccrma.stanford.edu/~jos/pasp/Soft_Clipping.html

Smith, J. O. (2010b). Freeverb. *Physical audio signal processing: for virtual musical instruments and audio effects* [Online book]. Retrieved from https://ccrma.stanford.edu/~jos/pasp/Soft_Clipping.html

Vetter, K. [katjav] (2013, January 15). Freeverb in vanilla Pd [Msg 4]. *Pure Data Forum~*. Retrieved from http://puredata.hurleur.com/sujet-7151-freeverb-vanilla

ACADEMIC VITA

Brian Fay
121 W. Fairmount Ave
Apt #2
State College PA 16801
bpf5050@psu.edu

Education :

B.S. in Information Sciences and Technology, Design and Development Option

Minor in Music Technology

The Pennsylvania State University, Spring 2014

Honors in Music

Experience:

(Summer 2013) Internship at the Applied Research Laboratory, Pennsylvania State University

(Fall 2012 – Spring 2013) Undergraduate Research Associate, Cyberinfrastructure Laboratory

(Summer 2012) IT Internship at QVC, West Chester PA

Activities:

(Fall 2012 – Spring 2013) Vice President of the Audio Engineering Society (Penn State Chapter)

Skills:

- General purpose programming: Java, C
- Web-specific programming: HTML, PHP, CSS, JavaScript
- Audio-specific tools: Pure Data, Max/MSP, SuperCollider, Logic
- Comfortable with Unix

Interests:

- Creative coding, maker culture
- Guitar (jazz, classical, and other styles)
- Electronic music production