I.INTRODUCTION

Convergence

Auto Gain and Dynamic Filtering Plugin for Guitar

Connor McCullough

Music Engineering and Technology

University of Miami

Coral Gables, United States

c.mccullough1@umiami.edu

Electronic music and heavy metal music for many years have been considered to be polar opposites. However, in recent years, the two genres have more and more begun to incorporate qualities of one another. Heavy metal which once thrived in a raw, under produced sound, has streamlined and cleaned its production in the 21st century, with the dominant style being highly accurately played and almost robot in nature. Extensive use of orchestral and choral synths, as well as programmed and bitcrushed drums and 808 bass drops have found there way into the genre. Meanwhile, electronic music has become increasingly aggressive, distorted, and given birth to many heavy subgenres, the latest being the dubstep trend and all of its respective subgenres. Many artists have already attempted crossovers, alternating heavy guitar with electronic passages or mixing styles of vocals. However, the one missing piece in creating a fully unified new genre is processing of electric guitar in the manner that these modern forms of electronic music are synthesized. In electronic music, the acoustic envelope is highly varied, and almost never similar to that of a guitar, which has peak amplitude and highest amounts of harmonic variance at the beginning, before decaying into a signal with energy concentrated at the fundamental frequency. The peak amplitude in an electronic “drop”, the equivalent of a particularly heavy low register riff on guitar, almost never is at the beginning of the signal. There is never a distinct transient in this style of electronic music, but instead peak amplitude that can occur at any point in the note. Another feature, which separates these synthesized signals from a distorted electric guitar, is highly varying resonant frequencies. These resonances usually correlate to the gain in some way and in this plugin, the two will be executed together using the sample envelope. As will be seen later, the processing of a highly distorted signal will present difficulties in that the traditional envelope of a guitar with a distinct transient is obscured to the point of being undetectable through traditional transient detection means.

II. TRANSIENT DETECTION AND ENVELOPE FOLLOWING

 The first stage of the plugin is a transient detection block. Implementation of the Auto Gain and shifting of the center frequency of the Reson Filter will begin when the transient detection function returns a value of 1. The concept of a transient refers specifically to the period of time in which a signal evolves quickly and unpredictably, usually as a result of excitation such as a hammer strike or pluck. The term *onset* is used to describe the single instant used to mark the transient, either at the start of the transient or at the earliest time at which the transient can be detected. One necessary step to determine the onset point of a signal is Reduction, subsampling an audio signal into a detection function (*A Tutorial on Onset Detection in Music Signals,* 1037). One of the most common methods of deriving a detection function is through following the envelope of a signal. This is done by half wave rectifying a signal and then using a low pass smoothing filter. After creating an envelope follower, the time derivative is often used to determine the point of onset. This allows sudden changes in energy to be represented as narrow peaks.

Figure 1: *Envelope of a snare drum (top), derivative of the envelope of a snare drum (middle), detected transients of the envelope (bottom).*

Figure 2: *Transients overlaid over the original signal (top), filtered waveform (bottom)*

 Another technique for determining presence of transients is looking at the frequency content of a signal. Energy increases linked to transients are usual a broadband event.

Signal energy is usually concentrated in low frequencies, so changes due to transients are often more noticeable at high frequencies (*Separtion of Transient Information in Musical Audio Using Multiresolution Analysis Techniques, A Hybrid Approach to Musical Note Onset Detection)*.

 The envelope follower method, with a few slight alterations, was used for this particular implementation of transient detection. First, a low pass filter was implemented by taking the FFT of the signal and cutting all frequencies below an arbitrary level. The magnitude was edited while the phase was preserved and the phasor was recreated before the Inverse FFT was performed to regain a signal in the time domain. The following is the code for the FFT implemented low pass filter:

tfft=abs(fft(sound));

theta=angle(fft(sound));

tfft2=tfft./max(tfft);

tsound=sound;

for loop2 = 11000:length(tfft)/2 % filter input

 tfft2(loop2)=0;

end

e=2.71828122846;

tfftnew=tfft2.\*e.^(1i\*theta);

tsound2=abs(ifft(tfftnew));

A half wave rectifier was then implemented by conditionally setting any levels below zero to zero.

for loop1 = 1 : length(sound); % Half wave rectify to complete envelope

 if tsound2(loop1) < 0

 tsound(loop1) = 0;

 else

 tsound(loop1) = tsound2(loop1);

 end

end

The derivative of the entire signal was then taken, and the function returned a value of one for any sample derivative that was greater than half the maximum or minimum derivative of the selected audio clip.

tsound=diff(tsound2);

M=max(tsound);

m=min(tsound);

save=1;

for loop3 = 1 : length(tsound);

 if tsound(loop3)>.5\*M && loop3-save>mindist

 tsound(loop3)=1;

 save=loop3;

 elseif tsound(loop3)<.5\*m && loop3-save>mindist

 tsound(loop3)=1;

 save=loop3;

 else

 tsound(loop3)=0;

 end

end

This value of half was determined to be the optimal value through trial and error, returning all transients in the snare drum sample without any false triggers. In a fully functional plugin, the sensitivity of the transient detection would be a parameter.

Problems arose in that the returned values often were very close together and multiple values of 1 were returned for one transient. This was solved by setting an arbitrary number of samples that the function has to wait before returning another transient. Since transients derivative values are often returned very close together, the number of samples that the computer has to wait can always remain small, no matter what the subdivision. Problems would not arise except for extremely rapid transient events. The function saves the last sample number to return a value of one and compares it to the current sample, determining if this number is greater than the selected wait time parameter described above. If the derivative is above half the maximum derivative for the sample but it occurs within the wait time, a value of zero will be returned.

 While initial testing of the transient detection for a snare drum went smoothly, transient detection for a heavily distorted guitar signal was not nearly as successful. Due to the highly varied and unpredictable nature of a very distorted guitar signal, the function was falsely triggered constantly so that the only times a transient was marked were after the wait period had expired. Many methods of transient detection call for splitting of the frequency spectrum into multiple bands and analyzing specific bands to detect transients*(Onset Detection in Musical Audio Signals)*. This seemed like the most logical method for modifying the function to process distorted guitar. In most genres of music, there are higher amounts of energy in the higher frequencies during the transient phase, with energy in the lower bands remaining fairly constant. By isolating these higher frequency bands, transients can be determined more accurately. However, when dealing with heavily distorted metal guitar signals, there is a constant wall of high frequency overtones that is independent of transient occurrences. This renders the method of frequency band isolation for transient detection ineffective in this particular instance. When comparing 10 filtered bands in MATLAB of a heavily distorted guitar signal, no particular frequency band had clearly visible transients in the waveform and the algorithm functioned equally as poorly in marking transient events. Due to the wide range in types of distortion and processing that the detection would have to accommodate, the most effective method of detecting transients is by processing a DI unprocessed guitar signal, or in the use of digital amp modeling plugins, placing this plugin or at least a transient detection plugin before any distortion plugins in the signal chain. Further understanding of the spectral content of extremely distorted guitar signals must be done before any particular solution can be found.

 The final stage of testing was done on a clean guitar signal. While the threshold for determining a transient had to be lowered from 50% of the maximum frequency to 30% of the maximum frequency, the function was still reliable and picked up all necessary transients. However, the need to change parameters depending on the instrument suggests that for a finished plugin, presets for each instrument may be necessary for optimal results, so that users don’t manually have to determine the optimum sensitivity parameters for each instrument.

III. RESON FILTER

A filter that resonates at any desired frequency can be created using a pair of complex poles, as opposed to a single pole, which can only be used to resonate at DC or Nyquist frequency. However, having two poles results in the resonating frequency not being exactly the angle θ of the pole. This is a result of the second pole shifting the peak. The true resonant frequency of an equation is given by the following equation, with Ψ being equal to the true frequency of resonance and θ being the angle of the complex pole:

cosΨ = 1+R2 cosθ

 2R

The actual design process of a resonant filter starts by choosing a bandwidth B and a resonant frequency Ψ. The pole radius R can then be derived from B using the following equation:

R = 1 – B/2

The next step is determining the cosine of the real pole angle θ using the pole radius and resonant frequency:

cosθ = 1+R2 cosΨ

 2R

The gain factor is then determined from the previously gathered information:

A0 = (1 – R2) sinθ

The final difference equation is as follows:

yt = A0xt + (2Rcosθ)yt-1 – R2yt-2

When the transfer function of this equation is derived using the Z-Transform, the matrix coefficient vectors are [A0] for the numerator and [1 -2Rcosθ R2] for the denominator. It is then easy to filter any signal using the MATLAB filter function, using the two matrix coefficients as values (*A Digital Signal Processing Primer,* 90-92*)*.

The primary focus in the implementation of this particular reson filter is in shifting the center frequency in terms of the envelope. In a completed version of a plugin, the parameters such as the gain factor and bandwidth will be variables as well, but for initial testing, A0, B (and as a result R) will be constant and the only variable will be Ψ.

Implementation of a filter with a dynamic frequency in MATLAB is much different than that of a filter with constant coefficients. The ‘filter’ function in MATLAB takes two vectors (numerator and denominator) with coefficients representing coefficients for each power of Z in the transfer function and returns the filtered audio signal. However, this function does not work for variables. Instead, the actual difference equation is used in a for loop, with the incremented variable being time. However, there have to be special conditional statements so that negative sample numbers aren’t accessed, which would cause a compiling error. In the Reson filter difference equation, samples that are delayed by both one and two samples are fed back. If the loop is accessing the first or second samples, a different difference equation is used that does not include these terms in order to avoid errors. This is implemented in MATLAB through a series of if and else statements (*Lab 5 Solution)*.

B = .0284; % 100 Hz in rad per sample

R = 1 - B/2;

psi = 1000\*pi/44100+4000\*pi/44100\*hanning(envlength);

theta = acos((2\*R/(1+R^2))\*cos(psi));

A = (1-R^2)\*sin(theta);

fsound = zeros(1,length(ftemp));

for loop6=1:length(ftemp)

 if (loop6 > 2)

 fsound(loop6) = A(loop6)\*ftemp(loop6) + 2\*R\*cos(theta(loop6))\*fsound(loop6-1) -R^2\*fsound(loop6-2);

 elseif (loop6 > 1)

 fsound(loop6) = A(loop6)\*ftemp(loop6) + 2\*R\*cos(theta(loop6))\*fsound(loop6-1);

 else

 fsound(loop6) = A(loop6)\*ftemp(loop6);

 end

end

end

IV. CALL FUNCTION

 While coding, it was determined that the most effective way of implementing the signal chain was through using a call function that essentially communicates between the transient detection and the gain and reson filter functions. The function loops through each sample of the signal of returned transients. It skips to the next sample if there is a zero, and if there is a 1, it first calls the gain function and processes it, and then calls the Filter function. These functions both process the input audio pre-transient detection. At the end of the function call, the loop is incremented by the number of samples that were processed by the filter and gain functions. If there is no function being called currently, the output audio is set to zero.

function [gsound] = AutoGain (sound,transient,envlength)

 for loop4 = 1:length(sound)-1

 if transient(loop4) == 1

 gtemp = Envelope(loop4,sound,envlength);

 glength = length(gtemp);

 gsound(loop4:loop4+glength-1) = gtemp

 if loop4 + envlength < length(sound);

 gtemp = ResonFilt(loop4,gsound,envlength);

 gsound(loop4:loop4+glength-1) = gtemp;

 end

 loop4 = loop4 + envlength;

 end

 end

end



Figure 3: *Hanning function used for the envelope.*

V. GRAPHICAL INTERFACE CREATION

 MATLAB allows creation of a graphical user interface through the GUIDE tool. Silders, text, buttons, and other features can be added to the window, which automatically creates functions in a .m file for each of the buttons. The functions can then be used for initializing slider values, or for mapping values in the plugin to movements of sliders or input values (*Creating a GUI with GUIDE)*.

The biggest consideration when creating a graphical interface for a plugin is user friendliness. Most importantly, this includes limiting parameter choices to only those that allow desirable operation, as well as using names and units that are intuitive to the user. The user interface for the Convergence plugin is divided into multiple sections based on the various operations of the plugin. The ‘general’ box includes a text box for entering a file name to process and begins processing audio, while the envelope length determines how long the processing of audio will take place following detection of a transient. This would need to be expressed in seconds or milliseconds as opposed to samples as these units are most intuitive to the user. The ‘detection’ box, which contains parameters related to transient detection, has sliders for sensitivity and wait time. Sensitivity would be expressed as a percent, correlating to the minimum and maximum possible useful ratios of the maximum detected derivative for detecting transients. Wait time would also be in seconds, and in a fully functioning interface, would be in some way related to tempo and subdivisions. The filtering box takes minimum and maximum frequency in Hz, as well as Q for bandwidth, as this is the most universally recognized parameter for bandwidth in equalizers. The gain box has a sensitivity slider, which multiplies the hanning window by a given number, causing the amplitude to be very low at the beginning and very high at its peak. This would also be expressed in the interface as a percent, with the range being all possible useful values to the user. The master gain would be implemented in the main of the code, but is present in the gain box as this is the most intuitive place for the user. This boosts the level of the entire signal by a given level in dB, in case the user feels the output is too quiet based on their other parameter choices.

Figure 3: *Designing of the graphic interface in MATLAB.*

VI. PRACTICAL APPLICATION CONSIDERATIONS

While the plugin is effective at processing pre-recorded audio, a different approach would be necessary when programming a plugin to process audio in real time. The biggest consideration would be the processing speed of the program and programming to maximize the efficiency of the program while not sacrificing its fidelity. While actual audio data should never be downsampled, there are many mathematical functions that can be downsampled without any audible difference. If a transient was detected every 10 samples, the maximum possible error would be about .2 milliseconds. This is certainly not an audible difference, however the problem arises when transient detection is used for muting such as in this plugin. A delay of 10 samples could create an audible pop in the signal. In this particular case, the transient detection should probably not be downsampled. The same applies for the Call Function, which should also not be downsampled. Because transients are represented only by unit impulses, they could easily be missed if any sample is skipped over. Calling a function late may also result in clicks and pops. However, in both the filtering and gain function, downsampling can be used liberally. The same value of the envelope can be applied to many samples at a time before the envelope value is incremented without any audible consequences. This could be tested by continuously lowering the sample rate until there are audible artifacts. A similar method could be applied to the sweep rate of the filter, where downsampling can also occur without altering what is heard by the listener.

 Another consideration in practical real time signal processing is the concept of causality. Causality in a system means that no future values can effect the present values (*Signals and Systems Using MATLAB,* 498). When prototyping in MATLAB using prerecorded samples, a program can function without being causal because any sample in the file can be accessed at any time once it is read in. However, when a steady stream of samples is being processed in a practical plugin, future sample values are unknown. This is not quite true in the absolute, because a small amount of delay can be introduced without audible consequences and audio can be processed in blocks. However, non-causal algorithms cannot be relied on extensively for plugins. The biggest weakness in this prototype in terms of causality is in the transient detection function. The function detects transients by finding the maximum derivative in the entire audio file and then comparing it to every other sample. This method becomes increasing unreliable as the size of audio blocks becoming processed gets smaller and smaller, which is necessary for real time processing. A frequency-based algorithm may need to be developed, but as mentioned above the frequency content of highly distorted guitar may decrease the reliability of such an algorithm. While some transient detection applications such as Beat Detection are more for post-production and don’t require causal real-time solutions, this particular application would have significant use in a live setting and would need to be causal. The best possible solution would be a comparison algorithm that initializes to zero until audio goes above a certain threshold. The first breach of the threshold is assumed to be a transient and the frequency content and envelope of this transient are saved in memory and compared to the current audio. If there are enough similarities, a transient would be detected and the envelope would be implemented. However, such a method will require much more understanding on the content of highly distorted guitar signals, as well as the types of changes in frequency and envelope that result from natural variations in picking or fingering.

VI. POSSIBLE IMPROVEMENTS

 While all intended aspects of the project were successfully implemented, the overall sound was off from what was intended and will require extensive testing of optimal parameters and additional debugging until it can be used in its original context of a modern electronic based filter for heavily distorted guitar. The reson filter took the majority of the body from the signal so the difference equation will have to be tweaked to decrease the Q factor and increase the bandwidth while still maintaining a distinct resonant sound. Also, the current implementation of the envelope results in an abrupt cutoff for any length of note. Ideally, the peak resonance will hold longer based on the length of the note, and eventually will gradually decay back into the unfiltered signal while waiting for the next transient. However, this comes back to the same problem of causality; that the filter cannot possibly know the full length of the note before it is processed and therefore cannot shape the envelope accordingly. Perhaps the biggest potential for this type of processing lies in manual envelope creation in the live setting, similar to that of a wah pedal, except applying a gain envelope along with resonant filtering. However, when notes are occurring very rapidly, the average human foot may not be able to accurately create an envelope fast enough to get the desired effect. Further research will occur to try to determine the most effective method in obtaining the desired effect.

REFERENCES

Duxbury, Chris, Mike Davies, and Mark Sandler. *A Hybrid Approach to Musical Note Onset Detection*. Proc. of 5th International Conference on Digital Audio Effects, Hamburg, Germany. London, UK: Queen Mary, University of London, 2002. 33-38. *Helmut-Schmidt-Universität*. Web. 3 Dec. 2012. <http://www2.hsu-hh.de/ant/dafx2002/papers/DAFX02\_Duxbury\_Sandler\_Davis\_note\_onset\_detection.pdf>.

Duxbury, Chris, Mike Davies, and Mark Sandler. *Separtion of Transient Information in Musical Audio Using Multiresolution Analysis Techniques*. Proc. of COST G-6 Conference on Digital Audio Effects, Limerick, Ireland. London,

UK: Queen Mary, University of London, 2001. 1-4. Web.

Pablo, Juan Pablo, Laurent Daudet, Samer Abdallah, Chris Duxbury, Mike Davies, and Mark B. Sandler. "A Tutorial on Onset Detection in Music Signals." *IEEE Transactions on Speech and Audio Processing* 13.5 (2005): 1035-046. *IEEE Xplore*. Web. 3 Dec. 2012. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1495485&tag=1>.

Hainsworth, Stephen, and Malcolm Macleod. "Onset Detection in Musical Audio Signals." Diss. Cambridge University, n.d. *Art & Science Laboratory*. Web. 3 Dec. 2012. <http://nagasm.org/ASL/icmc2003/closed/CR1047.PDF>.

Steiglitz, Kenneth. *A Digital Signal Processing Primer: With Applications to Digital Audio and Computer Music*. Menlo Park, Ca. [etc.: Addison-Wesley, 1996. Print.

Pirkle, Will. *Writing Audio Effect Plug-Ins in C++*. N.p.: n.p., 2012. Print.

*Creating a GUI with GUIDE*. MathWorks, n.d. Online.

Chaparro, Luis F. *Signals and Systems Using MATLAB*. Burlington, MA: Academic, 2011. Print.

Whitney, Jordan. "Lab 5 Solution." *MMI 502*. Colby Leider, n.d. Web. 5 Dec. 2012. <502rocks.pbworks.com>.