Multidist – The Distortion Unit with Optional Multiband Functionality

Michael Back 460409234

Final Report for DESC9115, 2016
Graduate Program in Audio and Acoustics
Faculty of Architecture, Design and Planning, University of Sydney

Abstract

Multidist is a new, exciting distortion effects processer developed by *Trentone*. The Multidist software offers users the freedom and variety that other processors do not. It has optional multiband capabilities, six separate distortion options, graphical user interfaces and it all runs in under 10 seconds! The software is evolving and it is shaping up to become very, very powerful

Introduction & Problem Description

Of all of the commercial DSP effects available, distortion is one of the most widely utilised. Most of the distortion is implemented in series with other effects, however. There are some units that focus on the splitting of a signal into multiple bands and distorting the bands separately, such as Ohmicide by Ohmforce (Ohmforce, 2016) and Trash 2 by iZotope (iZotope, 2016), but these DSP distortions are limited to a number of 4 frequency bands, and are expensive.

Multidist is a DSP effect that gives the user total freedom over their distortion. Unlike its competitors, Multidist gives the user options about whether or not to run the distortion as a multiband at all and gives a total of 8 distortion types, with discrete controls for gain and mix on each frequency band.

This report aims to serve as a discussion of the distortion unit, focusing on the specifications, the implementation and evaluation of the system as it stands. There will also be suggestions made for future releases of Multidist, and a discussion on the reality of implementing these suggestions.

Specifications

Multi-Band Splitting

When the main function is run, it prompts the user, asking if they want the project to run in multi-band or not. When yes is selected, the audio is prepped for multiband splitting. This is done by creating four discrete filters. The filters are created using MATLAB's filter designer. The filters used for each band are as follows. Butterworth for the Low Pass, Chebychev II for

the Low Mid Band, Butterworth for the High Mid Band and an undefined IIR for the High Frequency band. A graph of the resulting filter curves can be seen in Figure 1. The resulting filters are a Low Pass, Low Mid, High Mid and High Pass, which can all have their bandwidth adjusted by the user.

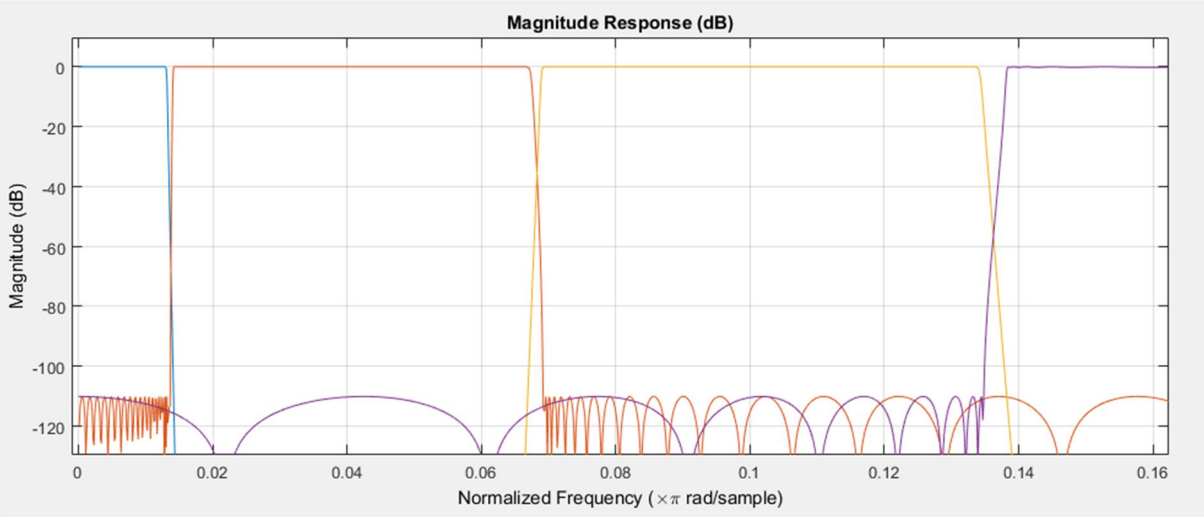


Figure 1 – Multi Band Splitting Filters

Next, the audio is filtered through these bands. A visual representation of the audio file filtered into discrete bands can be seen in Figure 2.

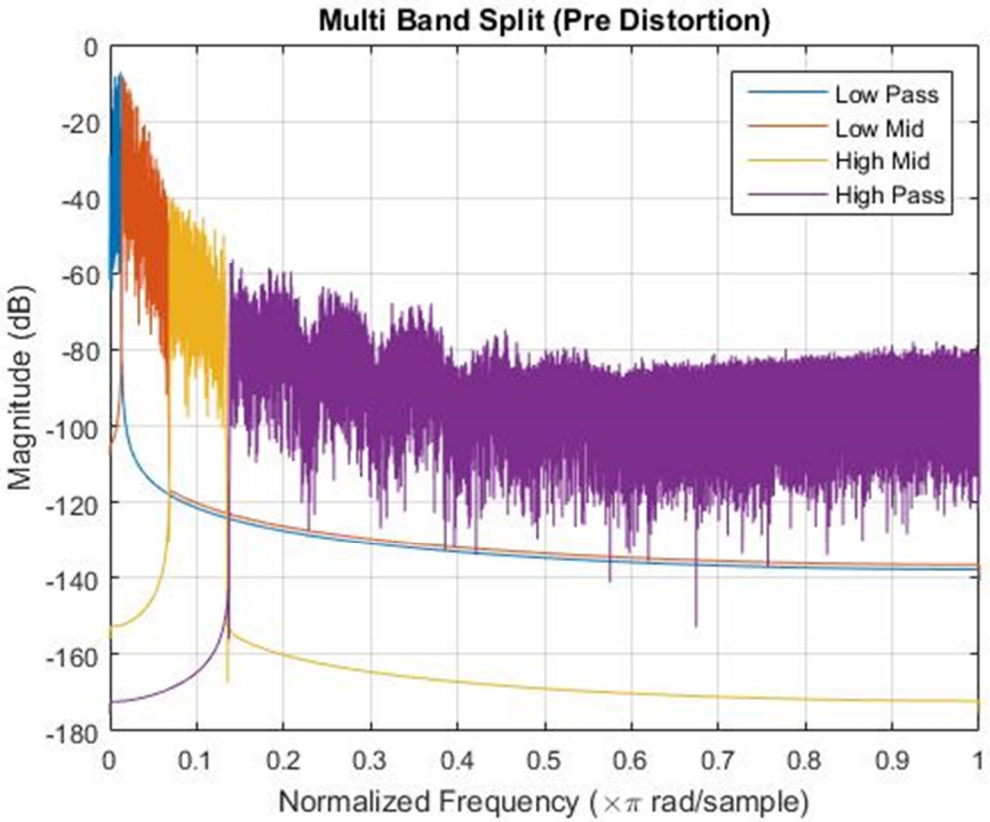


Figure 2 – Multi Band Split of Audio Input

Each of the resulting outputs is then run to a distortion function of the user's choice. Alternatively, if the user selects no for the multi-band prompt, the process skips this step and goes straight to the next part, distortion.

Distortion

The finalised product utilises a number of functions, six of which are distortion functions. These are the most important functions, and it's important to take a look at how they work. The distortion options, along with a brief description, are listed below. All the distortion options have 2 settings which are defined in a Graphical User Interface, being gain and mix.

1. ***Inverse Tangent 1***

   The Inverse tangent (or arctan) function uses the built in function *atan(x)* in MATLAB to manipulate the audio signal. The input is multiplied by the gain value and run through the arctan function.

   Mathematically speaking, inverse tangent can be defined by the following formula.

$$tan^{-1}(x) = \frac{i}{2}\log(\frac{i+x}{i-x})$$

   Essentially, with lower gain values, it will produce a soft clipping, and with higher gain values, the result will be hardclipping, which creates the distortion. A visual representation of the arctan function can be seen below in Figure 3. An example of this distortion can be heard in *invtan.wav*

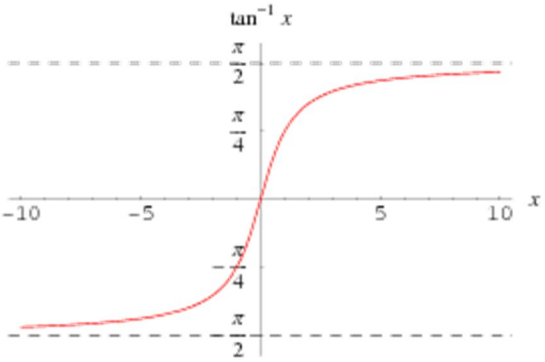


Fig. 3 - Inverse Tangent Plotted using the Real Axis (Weisstein, n.d)

2. ***Inverse Tangent 2***

   Similar to the first Inverse Tangent function, this function also uses a derivative of the Inverse Tangent. This one differs, however, by using the four-quadrant tangent inverse, represented by the built in function *atan2* in MATLAB. The four quadrant inverse tangent returns values in a range of –pi to pi, which is double that of the original inverse tangent (-pi/2 to pi/2) (Mathworks, n.d). This results in very harsh sounding distortion. This distortion could be great, but requires a lot of cleaning up in the function for it to be utilised. An example of this distortion can be heard in *invtan2.wav.* Be warned, it is harsh

3. *Fuzz/Distortion*

Fuzz/Distortion uses a non-linear exponential function to simulate hardclipping. It can be represented using the following equation.

$$f(x) = \frac{x}{|x|}(1 - e^{\alpha x^2/|x|})$$

Where alpha is the gain value. (Cardiff University, 2013) This essentially uses math to overdrive the signal to the point that it distorts, squaring off the tops of the waveform. A visual representation between overdrive and distortion using a sine wave can be seen below in Figure 4. An example of this distortion can be heard in *fuzz.wav*

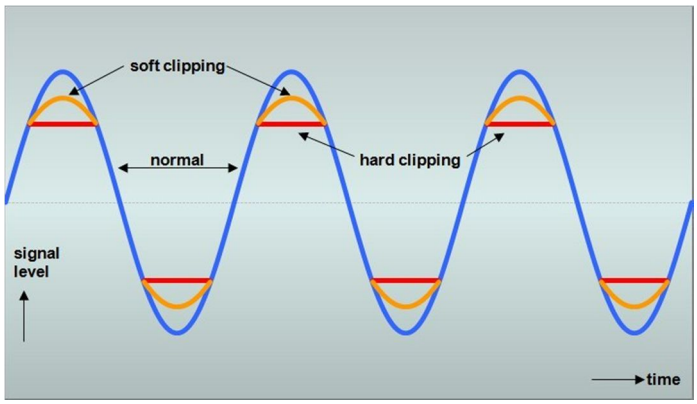


Fig. 4 – Soft Clipping and Hard Clipping (GoofyDawg, 2009)

4. *Hyperbolic Tangent*

The Hyperbolic Tangent function uses the in-built function *tanh(x)* in MATLAB. Basically, it will take every value of x (being the input) and output the hyperbolic tangent of the values. The curve of a hyperbolic tangent is displayed below in Figure 5.

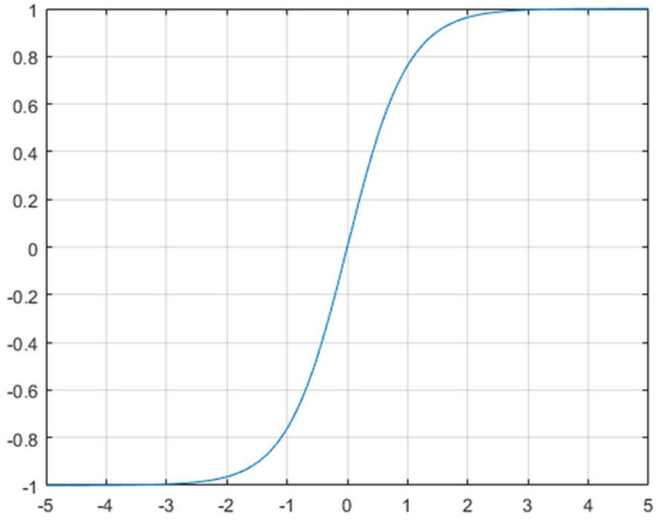


Fig. 5 – Hyperbolic Tangent from $-5 \leq x \leq 5$ (Mathworks, n.d)

As can be seen above, the curve is very similar, but not exactly the same to the one generated by the Inverse Tangent Function. An example of this distortion can be heard in *hyptan.wav*

5. ***Cubic Distortion***

   The way that cubic distortion was implemented was by experimenting with creating a cubic function, then running the audio through the function. The final function is as follows.

$$f(x) = \alpha x^3 + 5x^2 + \alpha x$$

   A visual representation of this can be seen below in Figure 6. Running audio through this distortion yields a very square sounding distortion. This distortion can be heard in *'cubic.wav'*

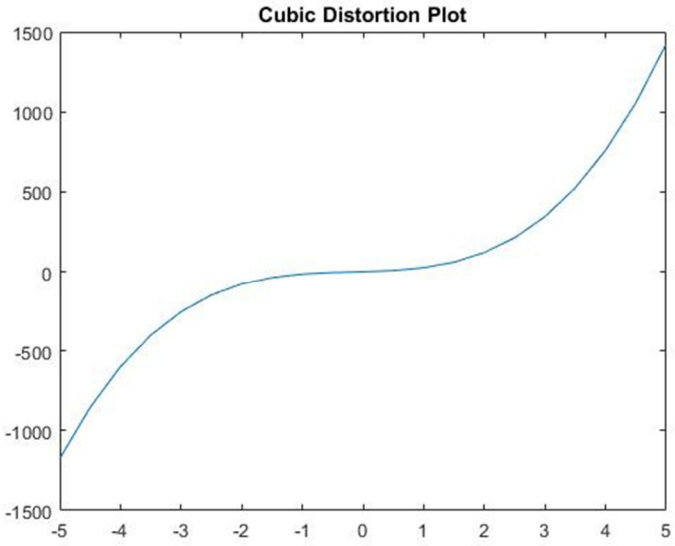


Fig. 6 – Cubic Distortion Plot

6. ***Cosine Distortion***

   This distortion function was more of an experiment than anything. What would happen if you ran audio through a cosine formula? To do this, the in-built function *cos(x)* is used. This just returns the cosine of the individual values in the input. The resulting signal can be heard in *cosine.wav*.

In all of these distortion functions, they have the option to be mixed with the original signal. This is done by having the following formula implemented into every distortion function.

$$output = (distortion \times mix) \times input(1 - mix)$$

Where mix is a value between 1 and 0. This means that the mix between distorted signal and dry signal will scale equally, whether it be 60/40, 50/50 or 100/0.

A list of all the functions and scripts used in this project can be seen in Appendix B.

Implementation

When looking under the hood at this distortion unit, there is a lot happening. The code, written and implemented in MATLAB, and, as stated earlier, the main script utilises separate functions to execute the final result

The unit reads an audio file (.wav), chosen by the user, after which the user is prompted to ask whether or not they want multiband distortion or not. Depending on the answer, two different graphical user interfaces can be executed; one for single band distortion and one for multiband. The simplified signal flow diagram for the system is shown below in Figure 7 (see Appendix A for a larger version of the diagram)

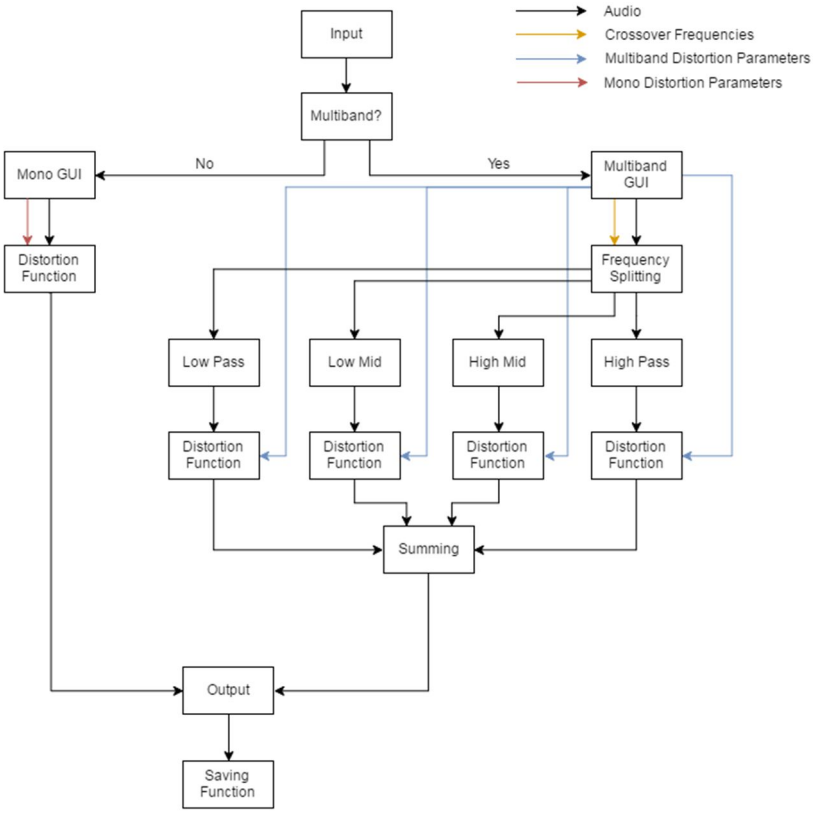


Fig. 7 – Simplified Signal Flow Diagram for Multidist

The parameters used to apply the effect on the signal are attained by the use of a graphical user interface (GUI). This interface was created using MATLAB's guide tool. There are separate interfaces for single band and multiband distortion. The parameters on the single band distortion GUI are very simple. There is a distortion type window, a gain slider and a mix slider. The values are implemented into the software when the user closes the window. An image of this GUI can be seen in Figure 7. (see Appendix C for a larger image)

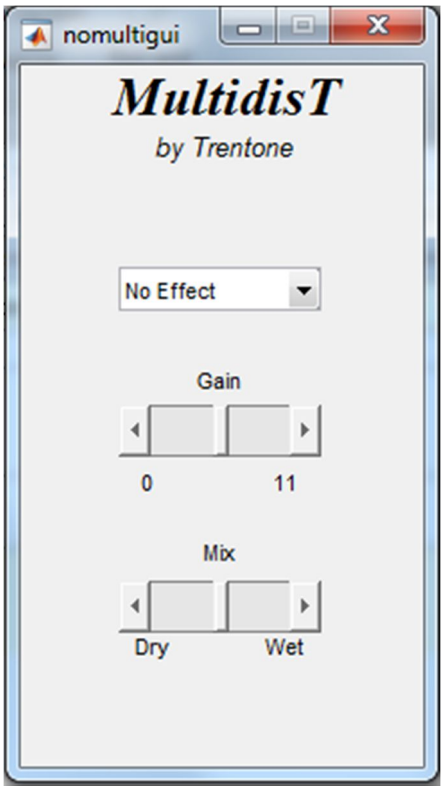


Fig. 7 – Single Band Distortion Parameter GUI

For the Multi-Band, the GUI is very similar to the single band, but the GUI has four effect choice menus, four sliders for gain and four sliders for mix, with each one being for their discrete frequency bands. A large image of this GUI can be seen in Appendix D.

After all the parameters are set, it will progress to the next function. For single band, it goes straight to the distortion function. For the multi-band, it goes to the splitting function, which was talked about above. After this, it then progresses to the distortion function, where the signal is distorted.

From here, for the single band, the audio is output. The multiple bands need to be summed before they can be output. There is a function used that will ask the user if they would like to save the file, and if they do, it will ask them to name it before saving it in the parent directory.

Evaluation

Multidist has been built in such a way that if anything goes wrong, the user will be notified. This does not mean, however that it is immune to any issues. The main point of weakness for Multidist would have to be some of the distortion functions. The user interface and multi-band splitting was done very in-depth with a lot of attention to detail. The distortion types serve as a way of experimentation, and as a result some of the distortions do not function as effectively as planned.

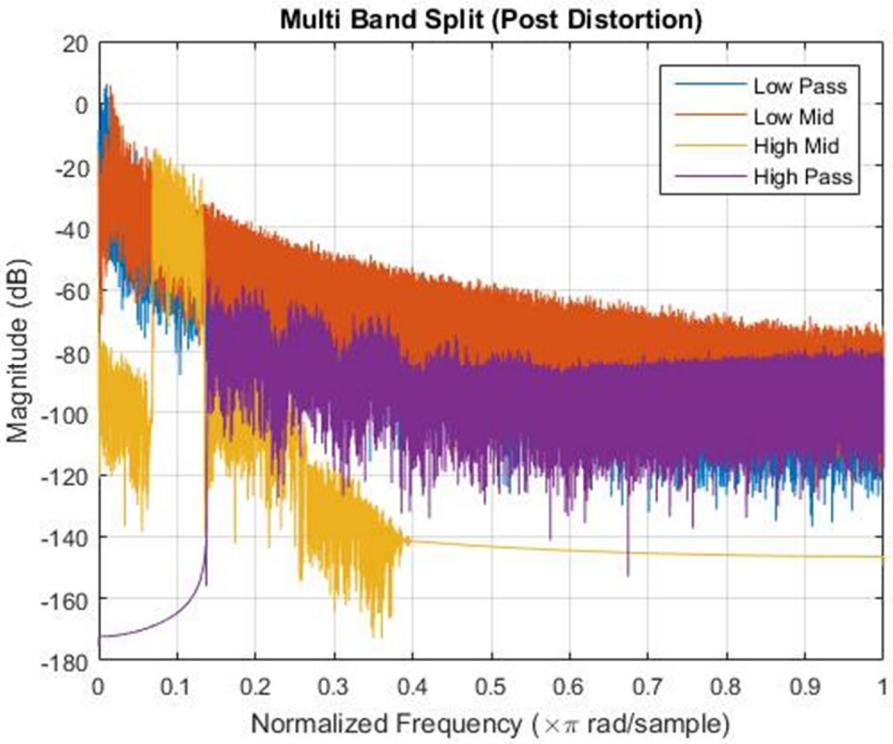


Figure 8 – Output of Separate Frequency Bands, graphed together

Figure 8 shows a graph of an input file after distortion, with each frequency band being coloured differently. This is a good indication that the product works. In this particular example, no distortion was used on the high frequency band. This can be seen due to the purple band being concealed under the low mid band.

In the future, improvements will be made in separate areas of the product. The distortion quality is something that needs looking at, with the potential for an endless number of distortions to be implemented, this product has the potential to become a very powerful unit, competing with the industry leaders.

To put this product ahead of the current industry leaders, the implementation of variable bands will be introduced. Users will be able to pick up to 10 or 15 separate frequency bands to distort independently. This is something that has not been seen on any multi-band distortion units as of yet. Of course, running Multidist in real time is a major goal.

A prototype GUI for realtime application can be seen in Appendix E.

Conclusion

Compared to other competitors in the market, Multidist stands up rather well. With implementation of future upgrades, it has potential to become the most powerful unit on the market. All that it needs is real time implementation and it will undoubtedly become a legendary bit of software.

References

Cardiff University (2013) Digital Audio Effects Retrieved from http://www.cs.cf.ac.uk/Dave/Multimedia_CM0340/PDF/06_Digital_Audio_Effects.pdf

GoofyDawg (2009) *Overdrive vs. Distortion | Guitar Gear* Retrieved from https://guitargear.org/2009/11/05/overdrive-vs-distortion/

iZotope (2016) *iZotope Trash 2 – Ultimate Distortion Plugin* Retrieved from https://www.izotope.com/en/products/create-and-design/trash.html

MathWorks (n.d) *Four Quadrant Inverse Tangent – MATLAB atan2* Retrieved from http://au.mathworks.com/help/matlab/ref/atan2.html

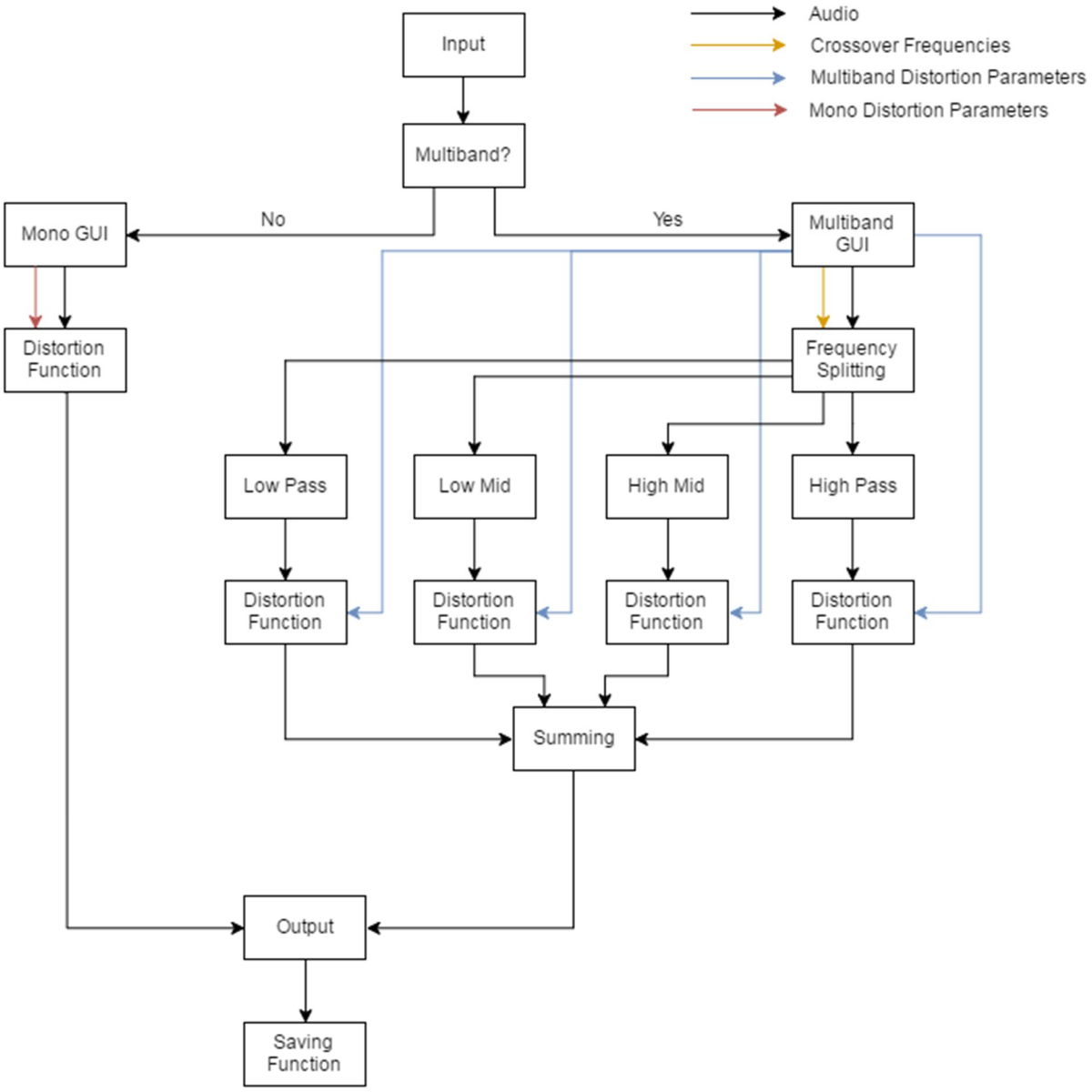
MathWorks (n.d) *Hyperbolic Tangent – MATLAB tanh* Retrieved from http://au.mathworks.com/help/matlab/ref/tanh.html

Ohmforce (2016) *Ohmicide – The Best Distortion Plugin on Earth* Retrieved from https://www.ohmforce.com/ViewProduct.do?p=Ohmicide

Weisstein, Eric W. (n.d) *Inverse Tangent. MathWorld--A Wolfram Web Resource* Retrieved from http://mathworld.wolfram.com/InverseTangent.html

Appendix A:

Signal flow diagram for Multidist.



Appendix B:

List of All Functions and scripts in Multidist

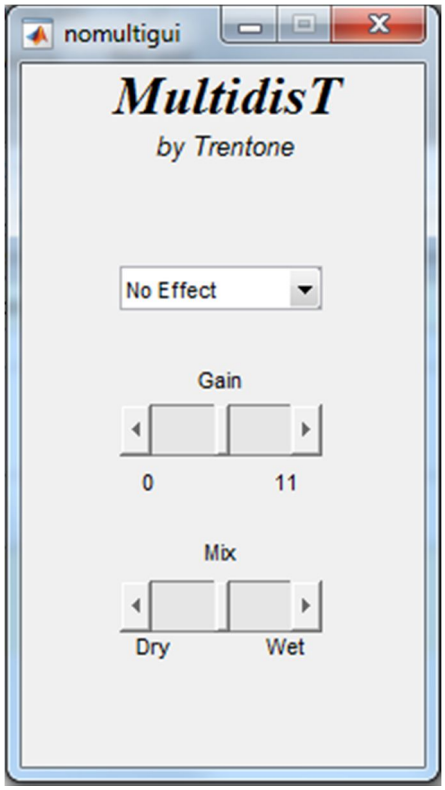
- Main_Script.m (This is the main script – RUN THIS)
- multiband_d.m (executes the audio manipulation process
    - freq_splitting.m (used to filter the audio into discrete bands)
    - audiofilt.m (filters input audio into the bands from freq_splitting.m)
    - apply_dist.m (used to apply distortion to each frequency band)
        - LPDist.m (used to determine & apply which distortion selected for LP)
        - LMDist.m (used to determine & apply which distortion selected for LM)
        - HMDist.m (used to determine & apply which distortion selected for HM)
        - HPDist.m (used to determine & apply which distortion selected for HP)
    - nomultidist.m (determines and applies distortion when running in single band)
    - plot_split.m (plots the frequency response of the bands)
- audiosave.m (function used to save file)

- Scripts
    - multidistbytt.m (script for multiband gui)
    - nomultigui.m (script for single band gui)
    - multiprompt.m (script for asking for multi or single band)
    - saveprompt.m (script for prompting for saving)

- Distortion Functions
    - cosine.m (function for cosine distortion)
    - cubic.m (function for cubic distortion)
    - distort.m (function for hyperbolic tangent distortion)
    - fuzz.m (function for fuzz/distrortion)
    - invtan2.m (function for inverse tangent 2 distortion)
    - od.m (function for inverse tangent distortion)

Appendix C:

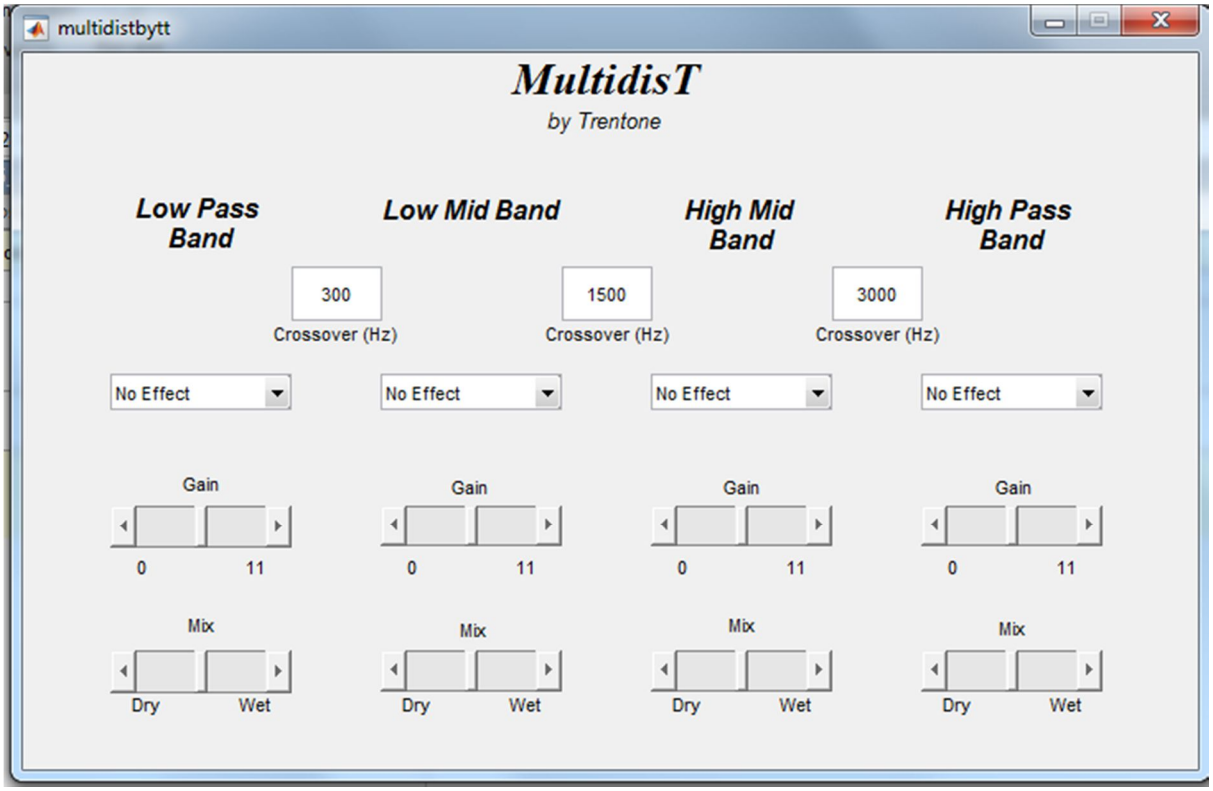
Images of currently implemented Graphical User Interface for Single Band Distortion



Single Band Distortion GUI

Appendix D:

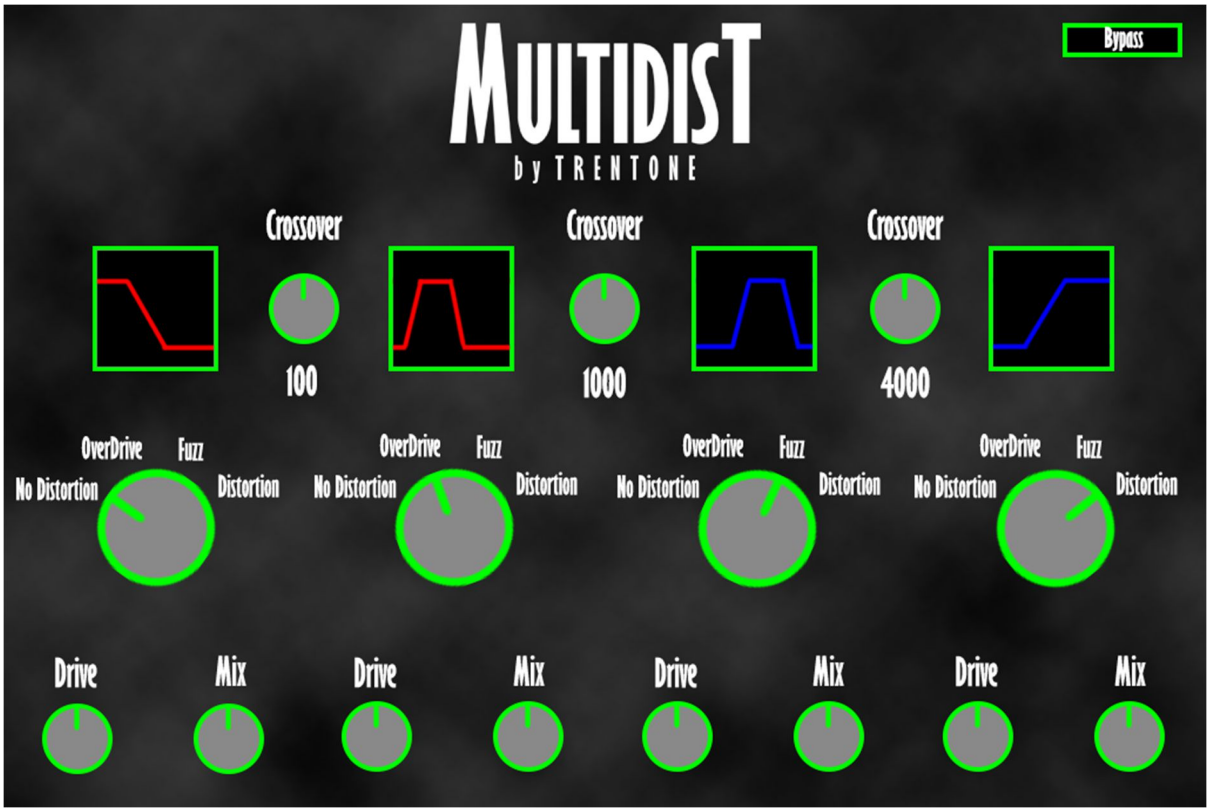
Images of currently implemented Graphical User Interface for Multi-Band Distortion



Multi-Band Distortion GUI

Appendix E:

Prototype Multidist GUI



Prototype GUI for Multidist by Trentone