

**TC70032E – Audio Programming II**

**Assessment 1: Novel Distortion VST3 Plugin**

May 2022

Harriet Rena Cerys Drury

21474551

# Abstract

This technical report highlights work undertaken to fulfil the assessment brief to create a novel VST3 Distortion effect using C++ and JUCE framework. Included is an analysis and breakdown of research materials to implement VST3 plugins, alongside signal processing theory for distortion types. A novel approach of focusing on creating a guitar distortion is picked, with research highlighting and influencing the design process.

Initial implementation of the distortion block is created in MATLAB, to generate graphical results and allow for analysis of distortion types. This is replicated in C++ via the JUCE framework to allow for fast implementation.

Auditory and visual results conclude the distortion works as intended and tested with guitar samples, it provided a satisfactory tone. It is noted that lower frequencies are affected by the distortion at a proportionally higher rate. This leads to the conclusion that future work would look at implementation of filters such as the band stop filter to control this.

Table of Contents

[Abstract 2](#_Toc104377267)

[1. Introduction 5](#_Toc104377268)

[Distortion Plugins 5](#_Toc104377269)

[Creating a VST3 Plugin 5](#_Toc104377270)

[An Overview of the Audio Process 7](#_Toc104377271)

[Soft Clipping 7](#_Toc104377272)

[Hard Clipping 9](#_Toc104377273)

[2. Methodology 11](#_Toc104377274)

[Designing the Distortion Stage 11](#_Toc104377275)

[MATLAB Prototype 12](#_Toc104377276)

[Testing 12](#_Toc104377277)

[Test Analysis 14](#_Toc104377278)

[Creation of a VST3 Plugin 14](#_Toc104377279)

[Program Overview 15](#_Toc104377280)

[PluginProcessor Source Code Design 16](#_Toc104377281)

[PluginEditor Source Code Design 18](#_Toc104377282)

[3. Results & Discussion 20](#_Toc104377283)

[Testing 20](#_Toc104377284)

[Future Work 21](#_Toc104377285)

[Conclusion 21](#_Toc104377286)

[4. Reference list 22](#_Toc104377287)

Table of Figures

[Figure 1.1 The Hierarchy of Audio Applications (Huang, 2022) 6](#_Toc104305605)

[Figure 1.2 The Audio Plugins Operational Connections (Pirkle, 2019, pp.24) 7](#_Toc104305606)

[Figure 1.3 The Waveform and Characteristic Curve of Cubic Distortion 9](#_Toc104305607)

[Figure 1.4 The Waveform and Characteristic Curve of Hard Clip Distortion 10](#_Toc104305608)

[Figure 2.1 A Block Diagram Breakdown of the Distortion Process 11](#_Toc104305609)

[Figure 2.2 A Breakdown of Signals from the Distortion Processing 11](#_Toc104305610)

[Figure 2.3 Max Soft Clipping, 0.5 Hard Clipping Threshold 12](#_Toc104305611)

[Figure 2.4 Max Soft Clipping, 0.3 Hard Clipping Threshold 13](#_Toc104305612)

[Figure 2.5 Soft Clipping at 0.3, 0.3 Hard Clipping Threshold 13](#_Toc104305613)

[Figure 2.6 Max Soft Clipping, 1.0 Hard Clipping Threshold 14](#_Toc104305614)

[Figure 2.7 The Creation of a VST3 Plugin in JUCE 15](#_Toc104305615)

[Figure 2.8 A Flowchart Overview of The Distortion Plugin 16](#_Toc104305616)

[Figure 2.9 The Declaration of Distortion Functions 16](#_Toc104305617)

[Figure 2.10 The Soft Clipping Cubic Distortion 17](#_Toc104305618)

[Figure 2.11 The Hard Clipping Distortion 17](#_Toc104305619)

[Figure 2.12 The Application of Distortion onto the Buffer Signal 17](#_Toc104305620)

[Figure 2.13 The processBlock of ‘PluginProcessor.cpp’ 18](#_Toc104305621)

[Figure 2.14 Gain Control 18](#_Toc104305622)

[Figure 2.15 The Creation of Tree State Parameters 19](#_Toc104305623)

[Figure 2.16 Creation of Distortion Parameters from The Tree State Values 19](#_Toc104305624)

[Figure 2.17 The Connected Graphic User Interface 19](#_Toc104305625)

[Figure 3.1 The Channel Visual Equaliser Without Distortion 20](#_Toc104305626)

[Figure 3.2 The Channel Visual Equaliser With Distortion 20](#_Toc104305627)

Table of Equations

[Equation 1 The Cubic Function (Tarr, 2019, pp. 160) 8](#_Toc104305628)

[Equation 2 The Branch Equation for Threshold Limiting Hard Clipping 9](#_Toc104305629)

[Equation 3 The Outputted Signal Equation 12](#_Toc104305630)

Table of Tables

[Table 1 Soft Clip Sigmoid Waveshaper Functions (Pirkle, 2019, pp.548) 8](#_Toc104305631)

# Introduction

This technical report highlights the research, implementation and final program to fulfil assessment criteria of creating a VST3 Distortion Plugin with the JUCE framework. The inclusion of contextual information of making a guitar focused distortion plugin is included to give the distortion focus, as well as inform decision making on parameters.

The structure of the report includes background information on audio plugins, an overview of the audio process including soft and hard clipping, prototyping, implementations, results and conclusions. The report aims to critically evaluate the created program with informed research and results.

## Distortion Plugins

For contextual placement, the focus of this distortion effect is to mimic the usability of a guitarist’s distortion pedal. Distortion is a common effect for the guitarists, with roots tracing back to when hardware components would be pushed to distort by breaking them (Russell, 2020) (Reiss and McPherson, 2015, pp.167–188). This led to terms such as overdrive, clipping and valve/tube distortion.

The usage of digital augmentation for this context is for researching clipping types and implementing a distortion plugin for the guitarist.

## Creating a VST3 Plugin

The specified form of this audio plugin is a VST3 plugin. This defines a set of audio plugin parameters to implement attributes and behaviours specifically for audio usage. Plugins are encapsulated C++ objects, using a set of base classes (Pirkle, 2019, pp.15). The audio plugin runs off of another application, to expand a host application function, shown with a red box in figure 1.1 below.

Diagram

Description automatically generated

Figure 1.1 The Hierarchy of Audio Applications (Huang, 2022)

Within the audio plugin interface, frameworks for creation such as JUCE exist (Pirkle, 2019, pp.15). The JUCE framework allows for audio programmers to work with defined classes to create both the user interface and digital signal processing of an audio plugin, standalone application and library. Figure 1.2 below highlights the connectivity of an audio plugin to an audio workstation. This is done within the project via the JUCE framework.

Diagram

Description automatically generated

Figure 1.2 The Audio Plugins Operational Connections (Pirkle, 2019, pp.24)

Utilisation of the JUCE framework within this distortion project allows for fast implementation of the user interface connecting to the audio processing. Prototypes of the distortion plugin are first generated within MATLAB. This is to check digital signal processing knowledge before moving into C++ via the JUCE ‘projucer’ to create a fully-fledged VST3 plugin.

## An Overview of the Audio Process

The audio processing for this plugin is focused mainly on the theory of distortion and parallel processing with additions of signals to create a result. For this distortion plugin, two types of distortion are used: the hard clip and soft clip distortions. Contextual information for guitar effect plugins is given also.

## Soft Clipping

Soft clip distortion is an effect where the amplitude of a signal is altered via a smoothing curve shaped by nonlinear processing models (Tarr, 2019, pp.159). Nonlinear processing of a signal in discrete time can lead to aliasing issues, caused by the expansion of a signals bandwidth (Julius Orion Smith, 2011) (Reiss and McPherson, 2015, pp.167).

There are many types of wave shaping functions for soft clipping, shown in table 1 below.

Table

Description automatically generated

Table 1 Soft Clip Sigmoid Waveshaper Functions (Pirkle, 2019, pp.548)

For this application, a cubic distortion is used, shown in equation 1 below.

Equation 1 The Cubic Function (Tarr, 2019, pp. 160)

The saturation of the signal along a smoothing curve allows for the implementation of a gain value within the cubic function. This value ‘a’ is also available to a user to implement as the amount of cubic distortion occurring. Zero would cause solely the x[n] signal be returned (no distortion), while one would allow for the whole cubic distortion to be factored into the signal (full distortion).

The smoothing of the cubic distortion can be shown with a sinewave with a frequency of two cycles per second. Figure 1.3 below shows the affected sine waveform and the characteristic curve of the cubic distortion. Full code for this can be found in appendix 1. The characteristic curve highlights how amplitudes are curved instead of cut. The sine wave form sees a reduction in amplitude at lower frequencies.

Chart, diagram

Description automatically generated

Figure 1.3 The Waveform and Characteristic Curve of Cubic Distortion

For this VST3 plugin, two cubic distortions are used in series, with separate knobs for distortion given to the user via a graphic user interface. When used within the context of a guitar effect plugin, the soft clipping is known as the overdrive (Malaker, 2019). An analogue example of an overdrive effect is the Ibanez Tube Screamer.

## Hard Clipping

Hard clipping is a distortion effect which limits the maximum amplitude of a signal based on the threshold programmed (Tarr, 2019, pp.157). The threshold is a changeable variable within this effect, which can be given to a user to change where amplitude is cut. If the amplitude of the signal goes over the specified signal, then it is cut to the threshold amplitude (Pirkle, 2019, pp.535). A branch equation for this is shown in equation 2 below.

Equation 2 The Branch Equation for Threshold Limiting Hard Clipping

Figure 1.4 highlights the characteristics of hard clipping on a sine wave with a frequency of two cycles a second. In comparison to figure 1.3 above, the threshold limit flattens the signal at lower frequencies, creating a harsher tone.

Chart

Description automatically generated

Figure 1.4 The Waveform and Characteristic Curve of Hard Clip Distortion

The application of the hard clip for the plugin is within parallel to the soft clipping distortions. The threshold is given within the graphic user interface and both resulting signals are added together with a final gain function after both. For a guitar effect plugin, hard clipping can be called distortion, and is famously used in analogue pedals such as the BOSS DS-2 (Malaker, 2019).

# Methodology

The methodology of this technical report highlights the implementation of a distortion VST3 plugin via JUCE framework.

## Designing the Distortion Stage

For this implementation, multiple distortion algorithms are used, with the aforementioned hard and soft clipping algorithms used. Figure 2.1 is a block diagram of the signal processing.

Diagram

Description automatically generated

Figure 2.1 A Block Diagram Breakdown of the Distortion Process

Where each block is a distortion effect. D1 signifies the soft clipping cubic distortion and D2 is hard clipping. From this diagram, we see that the signal is affected by cascaded effects. The signal is duplicated and processed with hard clipping while the soft clipping stage is occurring. After the signals are combined, a gain control is added to control the distortion amplitude. Figure 2.1 is a breakdown of the signals at each distortion stage.

Diagram

Description automatically generated

Figure 2.2 A Breakdown of Signals from the Distortion Processing

Where ‘g’ is the gain control, b(n) is the affected soft clipping signal and c(n) is the affected hard clipping signal.

From figure 2.2 above, an equation to find the output, is observed. This is shown in equation 3.

Equation 3 The Outputted Signal Equation

The design of the distortion blocks also allows for the changeable parameters to be implemented. From equation 1, we observe that ‘a’ is a parameter for amount of distortion on the soft clipping cubic distortion. Equation 2 highlights the threshold (thresh) of the hard clipping can be changeable.

## MATLAB Prototype

The digital signal processing for the distortion theorised above is primarily prototyped in MATLAB. This allows for the removal of graphic user interface concerns and fast programming implementation. In comparison, C++ has fast runtimes and does not provide limits to parameters and variables. Therefore, a MATLAB prototype can allow for focus on the signal processing, rather than programming. Once the MATLAB prototype is complete, implementing in C++ can focus on the programming and providing safe, reliable code.

The MATLAB prototype implements both distortion types and the gain control with the aforementioned structure. Variables for soft and hard clipping amounts are included, as well as a gain function limited to between 0 – 0.99 to avoid a feedback loop. This code closely follows implementation by Eric Tarr (2019, pp.147–181). Full code can be found in appendix 2.

## Testing

A sine wave is generated and passed through the signal processing. Figures 2.3 – 2.6 are plots of the distorted and original wave form, along with a characteristic curve. All plots utilise the gain value at 0.9 for ease of comparison. Both soft clipping stages use the same value also.

Chart, diagram

Description automatically generated

Figure 2.3 Max Soft Clipping, 0.5 Hard Clipping Threshold

Chart

Description automatically generated

Figure 2.4 Max Soft Clipping, 0.3 Hard Clipping Threshold

Chart

Description automatically generated

Figure 2.5 Soft Clipping at 0.3, 0.3 Hard Clipping Threshold

Chart

Description automatically generated

Figure 2.6 Max Soft Clipping, 1.0 Hard Clipping Threshold

## Test Analysis

From the above test code and resulting figures, the characteristics of the distortion process can be seen. Utilising a sine wave allows for the characteristic curves of different variables to be compared. From figures 2.3 – 2.6 above, it is noted that max soft clipping leads to the hard clipping threshold to impact the signal more notably. Values between the input and output signals in figures 2.3 and 2.4 are largely different to those with lower clipping, such as figure 2.5 where it was set to 0.3.

The tests also show that lower frequencies are further affected by these distortions. This could therefore create a lot of low frequencies which could need filtering. While beyond the scope of this project, it is noted that a high pass filter or band pass filter could be utilised for further control of the distortion.

## Creation of a VST3 Plugin

By utilsing the JUCE framework, the creation of a distortion VST3 plugin an occur via the creation of dependency files within the ‘projucer’ JUCE user interface. This is shown in figure 2.7.

Graphical user interface

Description automatically generated

Figure 2.7 The Creation of a VST3 Plugin in JUCE

This framework creates the required prerequisites for both a standalone and plugin version of generated code. This generates C++ code to obtain audio input and output, creating a buffer to store audio data, which can be accessed to create the distortion effect and creates a graphic user interface for editing.

The files created that can be edited are two header files (.h) and two coding files (.cpp). The ‘PluginProcessor.cpp’ and ‘PluginProcessor.h’ files are used to write the signal processing section of the code, with created functions making a baseline to build upon. ‘PluginEditor.cpp and PluginEditor.h’ are used for the input from users and for the graphical user interface. These files have dependency on the JUCE modules, which includes groups of header and coding files to create the framework.

## Program Overview

While dependencies on JUCE generated code is not fully explained, the process of creating the distortion is focused upon in this report. Figure 2.8 provides a flowchart for the project.

Diagram

Description automatically generated

Figure 2.8 A Flowchart Overview of The Distortion Plugin

To achieve this, the distortion types are first created as separate functions, to be called within ‘PluginProcessor.cpp’.

## PluginProcessor Source Code Design

Initially, all the required functions are declared within ‘PluginProcessor.h’, these are shown in figure 2.9 below. They are included in the ‘private’ portion of the class.

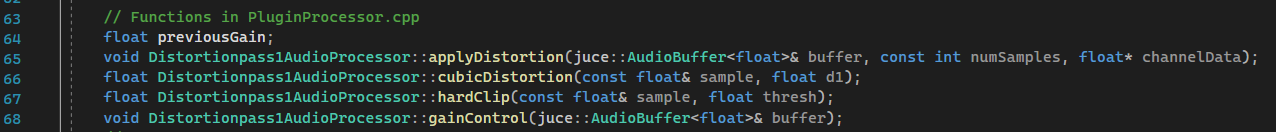


Figure 2.9 The Declaration of Distortion Functions

To add the distortion effect shown in equation 3, the hard clip and soft clip distortions are written within ‘PluginProcessor.cpp’. Figure 2.10 is the code for the soft clip (cubicDistortion) and figure 2.11 is code for the hard clip.

Text

Description automatically generated

Figure 2.10 The Soft Clipping Cubic Distortion

Text

Description automatically generated

Figure 2.11 The Hard Clipping Distortion

To minimise the code within one section, the generation of equation 3 is completed via another function named applyDistortion(). A benefit to this is the ease of changing the distortion block to what is required by the programmer. Figure 2.12 is the applyDistortion() function. This function also works with the user interface code to obtain values.

Text

Description automatically generated

Figure 2.12 The Application of Distortion onto the Buffer Signal

The function ‘processBlock()’ predefined by JUCE is where the applyDistortion() function is called. This includes working with the aforementioned buffer for real time application and working between data from input channels. Figure 2.13 highlights the syntax used within the processBlock function and therefore how distortion is added to an inputted signal.

Text

Description automatically generated

Figure 2.13 The processBlock of ‘PluginProcessor.cpp’

It is noted that gain control is also added to the distortion outside of the applyDistortion() user defined function. This is to utilise the pre-existing JUCE framework for gain control, built into a user defined function (figure 2.14).

Text

Description automatically generated

Figure 2.14 Gain Control

The final output of this code is the same as the original prototype within MATLAB. The next portion of code is the user interface and obtaining values from the user.

## PluginEditor Source Code Design

The focus of this report is largely the distortion effect and implementation for a VST3 Plugin. However, within PluginEditor, a basic user interface is created.

To input the user’s parameters into the signal processing code above, usage of a JUCE tree state (juce::AudioProcessorValueTreeState()) occurs. This stores user data and can be accessed via the ‘PluginProcessor.cpp’ code. Figure 2.15 is the creation of the tree state values for distortion and gain controls within ‘PluginEditor.h’ and figure 2.16 is ‘PluginProcessor.cpp’ accessing the values.

Text

Description automatically generated

Figure 2.15 The Creation of Tree State Parameters

Graphical user interface, text

Description automatically generated

Figure 2.16 Creation of Distortion Parameters from The Tree State Values

These tree states link the two parts of source files together and allow for immediate change of values within the distortion processing. Once these parameters are defined, controls for object placement, style and movement are implemented within ‘PluginEditor.cpp’.

The source files implemented creates a graphic user interface (figure ?.?) with functional digital signal processing as a VST3 plugin. This can further be improved upon and edited as required.

Graphical user interface

Description automatically generated with medium confidence

Figure 2.17 The Connected Graphic User Interface

# Results & Discussion

## Testing

Previous testing in section 2 highlights the prototype distortion function testing and how characteristics of a sine wave are changed. From this we can conclude that lower frequencies are highly changed or distorted. From an auditory assessment, this can be heard when using the VST3 plugin with a digital audio workstation. This can also be seen with the visual equaliser in Ableton Live. Figure 3.1 is a guitar track without distortion and figure 3.2 is the same guitar track with distortion added.

Graphical user interface

Description automatically generated

Figure 3.1 The Channel Visual Equaliser Without Distortion

Graphical user interface, chart

Description automatically generated

Figure 3.2 The Channel Visual Equaliser With Distortion

With both qualitative and quantitative data suggesting these outcomes, it is supported that the distortion implemented has a larger effect on low frequencies in comparison to middle and high frequencies.

Further testing, guitar audio is placed into the distortion and various thresholds and distortion amounts were tested, alongside the gain control. It is concluded that the auditory response is similar to a guitar pedal such as the Boss DS-2. It is concluded that the distortion works as expected.

## Future Work

This distortion pedal employs fundamental signal processing theory within its actualisation. Therefore, the ability to build upon the work is big. As previously discussed, a big downside is lack of tone control to employ a filter to disrupt all the low frequencies created within the distortion. Therefore, future work would look at filtering pre and post distortion for maximal gains. Furthermore, the distortions used are audibly small in nature. Further research into wave shaping with rectifiers would build upon this and create new effects.

For the distortion effect to be fully realised as a guitar pedal digitisation, there may also be further research into the differences between clipping and saturating a signal.

## Conclusion

To conclude, the above technical report discusses the theories behind distortions used within a guitar pedal framework. This is with alignment to the set coursework brief and allows for contextual displays of information alongside the digital signal processing and programming. The final design is implemented first as a prototype in MATLAB before moving into JUCE framework to create a VST3 plugin within the C++ language.

The resulting program contains hard distortion, soft distortion and gain control within a generated graphic user interface connecting to the signal processing. This signal processing occurs in real time, with the usage of a buffer created by JUCE framework to allow for how reloads and immediate responses.

It is concluded via testing and auditory conclusions that the distortion works as expected, with low end frequencies being generated at a higher rate. This leads to the conclusion that inclusion of filters would provide more control over the overall tone. It is also concluded that it sounds similar to guitar plugin effects.

# Reference list

Huang, Q. (2022). *Audio Programming 2. (Week 9) JUCE in Audio Programming*. [Lecture Notes Distributed in TC70032E – Audio Programming II. University of West London].

Julius Orion Smith (2011). *Physical Audio Signal Processing : for Virtual Musical Instruments and Audio Effects*. Lexington: W3k Publishing.

Malaker, E. (2019). *Soft Clipping Vs Hard Clipping – What Is the Difference?* [online] Humbucker Soup. Available at: <https://humbuckersoup.com/soft-clipping-vs-hard-clipping-difference/> [Accessed 17 May 2022].

Pirkle, W.C. (2019). *Designing Audio Effect Plugins in C++ : for AAX, AU, and VST3 with DSP Theory*. New York: Routledge, pp.15–30, 535–563.

Reiss, J.D. and McPherson, A.P. (2015). *Audio Effects theory, Implementation and Application*. London: Taylor & Francis Group, pp.167–188.

Russell, A. (2020). *The Complete Guide to Distortion and Saturation*. [online] EDMProd. Available at: <https://www.edmprod.com/distortion-saturation-guide/> [Accessed 24 May 2022].

Tarr, E. (2019). *Hack Audio: an Introduction to Computer Programming and Digital Signal Processing in MATLAB*. London: Routledge, pp.147–181.

# Appendix 1. The Distortion Effects with Plotting



Functions:



# Appendix 2. The Prototyped Distortion



Functions:

