1 Project work coding JUCE Vst plugin

1.1 Introduction

In this project, I undertook the creation of a simple yet foundational VST (Virtual Studio Technology) plugin using the JUCE framework, with C++ as the coding language within the Visual Studio IDE. This endeavor aimed to provide a thorough understanding of VST plugin development while crafting a basic volume fader plugin as a springboard for future, more intricate projects.

1.2 VST Plugin Overview

VST (Virtual Studio Technology) plugins are software components used within digital audio workstations (DAWs) to extend their functionality and provide additional audio processing capabilities. These plugins can range from simple effects processors to complex virtual instruments, offering users a vast array of tools for music production, mixing, and sound design. VST plugins are widely utilized by musicians, producers, and audio engineers to enhance creativity, streamline workflows, and achieve professional-quality results within digital audio environments.

1.3 Project Goal

The primary objective was to delve into the intricacies of VST plugin development, utilizing the JUCE framework and C language, to construct a functional volume fader plugin. This plugin serves as a cornerstone for expanding expertise in crafting diverse VSTs, ultimately enriching capabilities in music production and mixing.

1.4 Motivation and Artistic Value

The motivation behind this project stemmed from a passion for the creative fusion of music and technology. By delving into VST plugin development, I sought to augment my toolkit for music production and audio engineering. Through the creation of bespoke plugins, I aimed to enhance artistic expression and refine techniques for manipulating audio within digital workspaces.

1.5 Tools and methods

JUCE, renowned for its versatility and robustness in audio application development, was chosen for its comprehensive feature set tailored specifically for building VST plugins. Coupled with the power and efficiency of the C++ programming language, JUCE provided an ideal platform for realizing the project objectives. Visual Studio, as the preferred IDE, offered seamless integration with JUCE, facilitating coding and debugging processes. FL Studio served as the designated DAW for testing the functionality of the VST plugin, providing a familiar environment for evaluating plugin performance.

1.6 Pros and Cons of Chosen Tools

The selection of JUCE and C offered several advantages, including comprehensive documentation, community support, and cross-platform compatibility. JUCE's intuitive design and extensive feature set streamlined development, while C's performance-oriented nature ensured optimal efficiency within the plugin. However, the learning curve associated with JUCE and C may present challenges for newcomers, necessitating dedicated time and effort to master these tools fully.

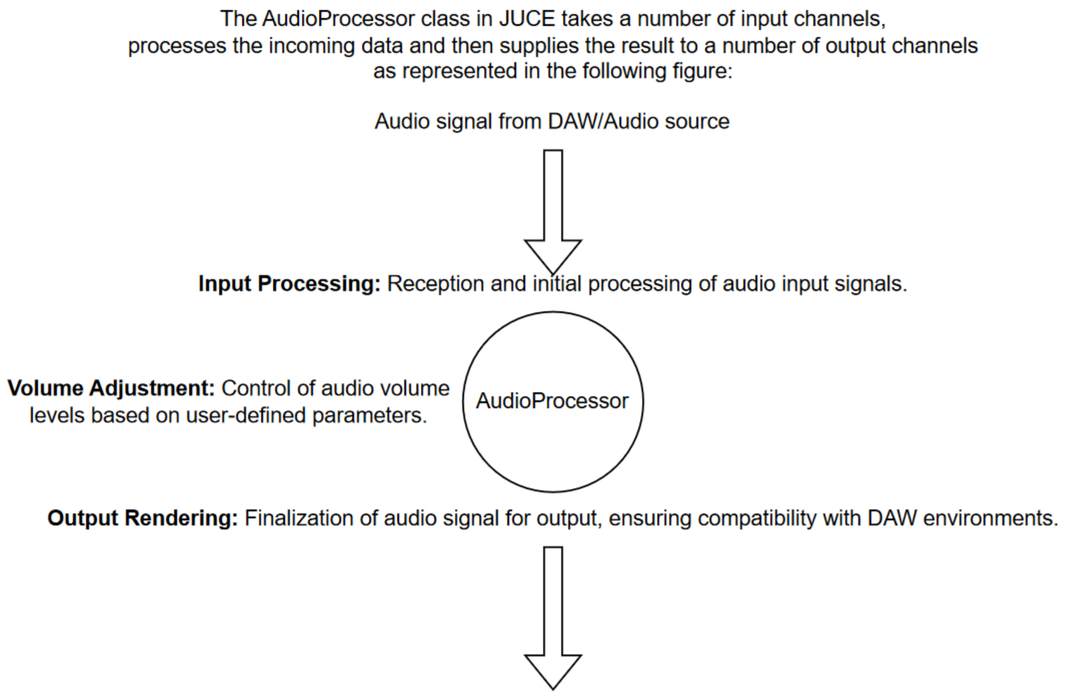
1.7 JUCE project files

JUCE provides several essential project files, each serving a distinct purpose in VST plugin development:

- PluginProcessor.h/cpp: These files define the core processing functionality of the plugin. The PluginProcessor class handles audio processing, parameter management, and communication with the host DAW.

- PluginEditor.h/cpp: Responsible for the graphical user interface (GUI) of the plugin, the PluginEditor class manages the visual representation of the plugin within the DAW, including parameter controls and display elements.

These template files serve as the foundation for building VST plugins using JUCE, providing a structured framework for efficient development and customization. Files with ending .h primarily contain declarations. They outline the structure and interface but do not contain the actual implementation code. Files that end with .cpp, on the other hand, contain the implementation code. They include the necessary headers, implement the member functions declared in .h, and define the behavior of the class. This file is where the actual functionality is coded, including audio processing algorithms, parameter handling, and interactions with the host DAW.


The AudioProcessor class in JUCE takes a number of input channels, processes the incoming data and then supplies the result to a number of output channels as represented in the following figure:
Audio signal from DAW/Audio source
Input Processing: Reception and initial processing of audio input signals.
Volume Adjustment: Control of audio volume levels based on user-defined parameters.
AudioProcessor
Output Rendering: Finalization of audio signal for output, ensuring compatibility with DAW environments.