System Design and Architecture

Umang Hirani 92200133025

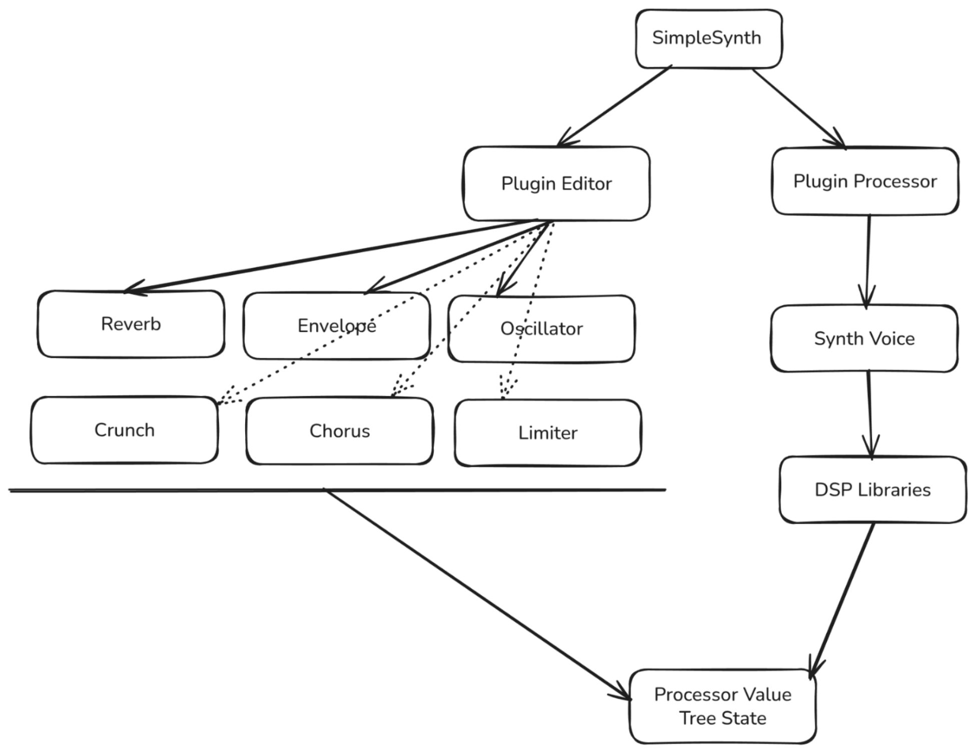
The System Design and Architecture of my Audio Synthesizer is as follows :-

The synthesizer was built on top of the JUCE Framework which is a C++ based framework that directly provides an interface for you to communicate with the audio server of any given OS or system and prevents you from reinventing the whole wheel from that functionality.

It also provides us with the UI toolkit to make the Editor UI for controlling all the parameters of the synthesizer and also many DSP and other libraries on top of that to change the way our synth behaves, both visually and audibly.

Here are some of its features as requested in the documentation :-

Modular Design:



The whole comprises of these components :-

Editor Components

These mainly are the components that comprise the UI when put together. From the sliders to the combo-boxes to the audio visualiser components, everything that the user sees and interacts with is made using the Plugin Editor Class which can be said as the frontend of the whole plugin.

It has all the methods implemented which are related to the look and feel of each component and their visual behaviour. Each component is made by extending the Component class and overriding and implementing some of its pure virtual methods.

There are listener classes for each of the components and elements so we can handle the changing of the knobs to the selection or even the resizing of the whole plugin window.

The sliders can be attached to the process-value-tree-state to then handle the updation of the actual processor value with respect to the user interface.

The Plugin Processor

This part can be stated as the backend of the whole plugin which handles majority if not all of the audio processing part and handles all the changes related to the change of any parameters of the actual plugin backend.

This is the part of the whole plugin which is integrated with all the DSP modules and libraries and also handles all the DSP logic. The whole flow of the sound being generated from the oscillator to it being passed through the envelope and then also it being passed through all the effects of the plugin and then hitting the limiter, this whole flow is being handled by the processor.

The Synth Voice Class

As a polyphonic synthesizer, the plugin has the ability to handle many voices at once, meaning it has to handle all the parameters of each and every voice that is being generated by the synth and also destroyed after they are stopped playing.

This mainly consists of all the low level implementations of how each and every event is handled in terms of MIDI and DSP. From handling the start of a note to the movement of the pitchwheel, all this logic is written in this class. Here all the DSP libraries are imported and then used to further modify and shape the sound as the user intends it to be.

Technology Stack:

Programming Language - C++ and CMake

The C++ language was a pretty obvious choice for this project due to it's performance and real time audio processing capabilities, making it far better and more viable as compared to the languages such as python and java.

Also I had considered languages like Rust or Go due to their raw speed or faster compilation but both get outshined by C++ in terms of builtin libraries for audio and the community support for audio development. Since all the legacy audio related plugins and softwares have been written in C++ if not C the ancestor of it, it was a pretty obvious choice to choose it over any other languages.

The build tool I used was CMake which is an exceptionally good but difficult to learn build tool for C++ related applications. This means that now I can compile my program efficiently and also don't have to depend on other dependencies such as keeping the files of the program that are used from other git repositories or downloading JUCE and other dependencies as it will handle downloading all that if that are not present in the current repository.

Framework - JUCE (Jules' Utility Class Extensions)

JUCE is an open source C++ framework which is used to build audio related applications which has been adopted by the likes of Netflix, ROLI and Modalics. It provides builtin functionalities to and classes and methods and libraries to make the UI of the plugin and also process all the incoming/generated audio.

Libraries

- Signalsmith's basics library
- Maximilian DSP library
- JUCE DSP library
- C++ STL library (for functions such as sinewave and stuff)

Scalability Planning:

Since this will be used on a local desktop only and not by many users concurrently, scalability is not an issue. However the performance is still an issue because the user may use many instances of this plugin concurrently and it should not make the computer lag by any means possible.

This can be achieved by properly handling the audio processing as that is occupying the majority of the computational workload.