Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Lengyel Dániel

FM SZINTETIZÁTOR ELKÉSZÍTÉSE JUCE FEJLESZTŐKÖRNYEZETBEN

BUDAPEST, 2020

Tartalomjegyzék

1 Bevezető

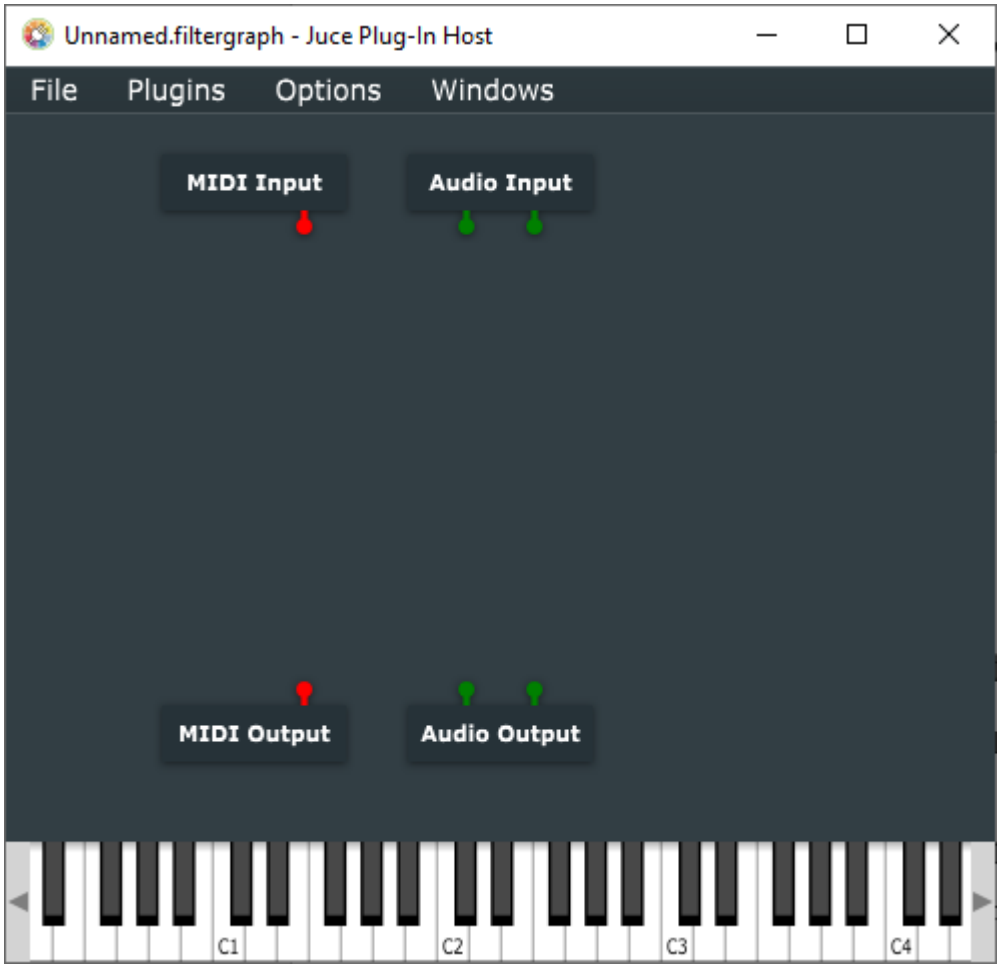
A féléves zenei jelfeldolgozás projektem témájaként egy FM szintetizátor elkészítését választottam. Kedvenc hobbim a zenélés, és szinte az összes számomban használok frekvenciamodulációs szintézist, így nagyon érdekelt, hogy pontosan milyen algoritmusok alapján működik. A tárgyat is azért vettem fel, mert sok különböző digitális szintetizátort használtam már, szeretek programozni és kíváncsi voltam, hogy mégis hogyan hozzák létre ezeket a szoftvereket.

Az egyik legelterjedtebb hangtechnikai alkalmazásfejlesztő keretrendszer a JUCE, amelyet a zeneiparban rengeteg cég felhasznál a szoftverjeikben. Ha valaki jártas a témában, bizonyára hallott már az M-Audio, Korg, Presonus vagy Arturia cégekről, akik mind igénybe veszik a JUCE szolgáltatásait. A JUCE használata mellett elengedhetetlen a Projucer, egy integrált fejlesztői környezet, amely segítségével könnyen kezelhetők a JUCE projektek. Nagyon egyszerűvé teszi a projektek futtatását különböző operációs rendszereken a kód változtatása nélkül [2]. A JUCE és Projucer 6.0.4-es verziójában, illetve Visual Studio 2019-ben dolgoztam. Ezeken kívül felhasználtam még a Maximilian könyvtárat, amely szintén sok hasznos előre elkészített függvényt is osztályt tartalmaz hangtechnikai programozók számára. A feladatom során a Maximilian-ból az oszcillátor osztályt használtam fel [1].

A követelmények egy két oszcillátoros szintetizátort írtak le, ahol az egyik modulálja a másikat és mindkét oszcillátor külön ADSR generátorral rendelkezik. Szinuszos hullámforma az alap, és lehet külön állítani a frekvenciaarányokat. A feladathoz még annyit adtam hozzá, hogy lehet állítani a frekvenciamodulálás mélységét, külön-külön az oszcillátorok hangerejét, illetve mindkét oszcillátor hullámformáján is lehet változtatni.

2 A program felépítése

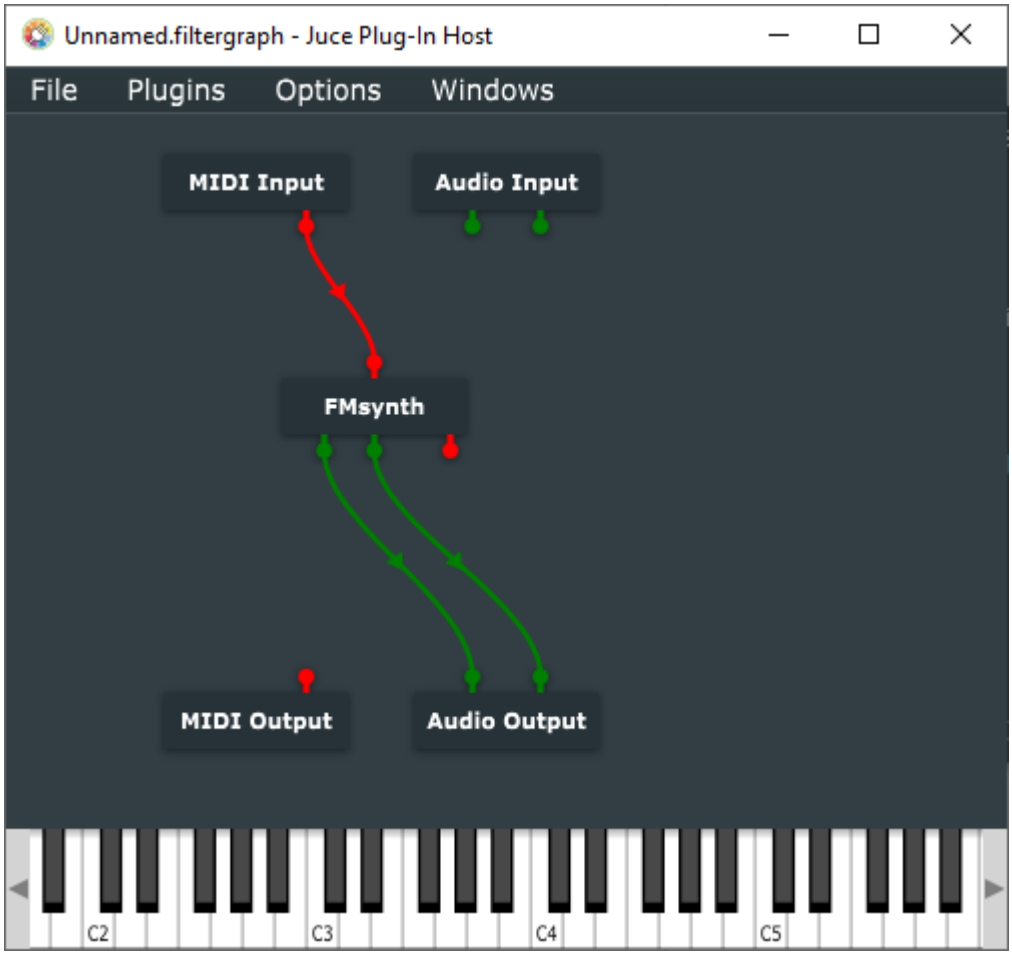
2.1 Kezdeti akadályok



2.1 A JUCE audio plugin host kezdeti állapota.

Szerencsére a JUCE elsajátításához számos hasznos forrást lehet találni az interneten, sok fórumbejegyzést és youtube videót megnéztem, amíg megértettem a működését. Nagyon megkönnyíti a hibakeresést a JUCE-szal csomagolt audio plugin host felhasználása. Ennek segítségével nem kell minden egyes alkalommal a program felépítésekor megnyitni egy DAW-t, és azon belül leteszteltelni a programot. Beállításához először a letöltött JUCE mappán belül, az extras, AudioPluginHost, Builds mappában meg kell nyitni azt a projekt fájlt, amilyen fejlesztői környezetben dolgozunk, az én esetemben ez a Visual Studio 2019 volt. Valószínűleg nem lesz előre felépítve a futtatható fájl, tehát a projektet meg kell nyitni, majd egyszer build-elni. Ez után az én

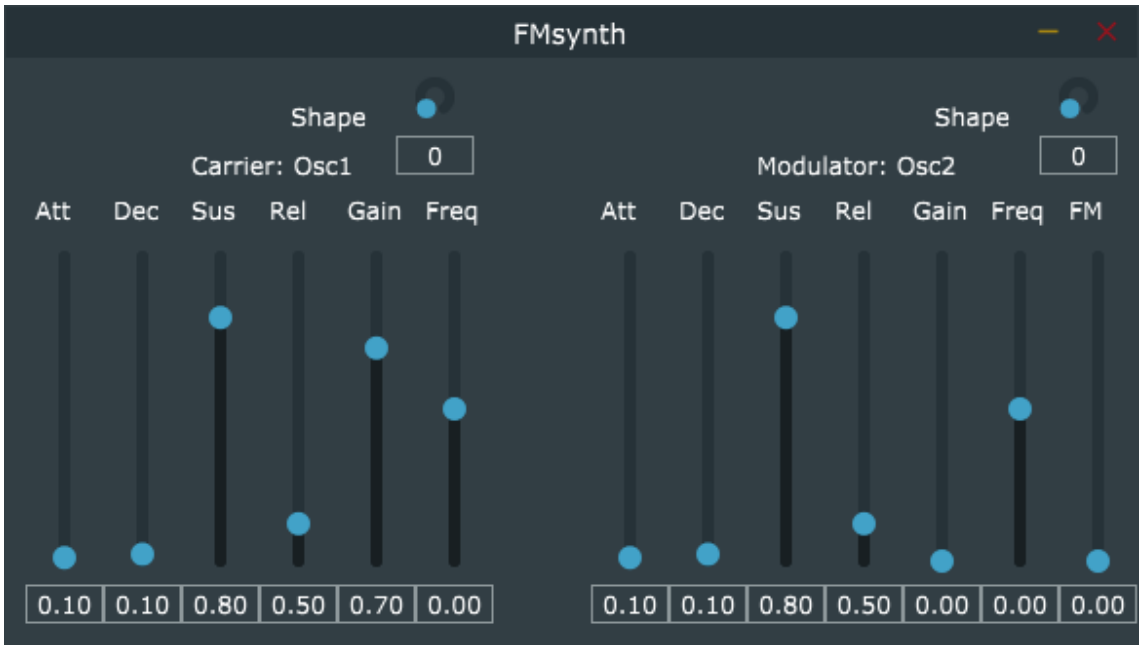
az x64, Debug, App mappában találtam meg az AudioPluginHost.exe fájlt, amit fel lehet használmi a VST plugin tesztelése során. Visual Studio 2019-ben, a VST projekt beállításaiban a Debugging menüpont alatt be kell állítani a Command szövegdobozban, hogy erre az .exe fájlra mutasson. A projekt futtatása után a **2.1**-es képen látható ablakot kell látni.



2.2 A JUCE audio plugin host a saját VST hozzáadása után.

Miután ezt megtörtént, az ablakban a következő menü pontok sorozatát kell követni: options, edit the list of available plugins, options, scan for new or updated VST3 plugins. A felugró ablakban ki kell választani a mappát, ahol található a projekt VST3 build-je. Ez után az audio host felületén jobb klikkelve felugrik egy menü, aminek alján lennie kell egy „yourcompany” feliratnak. Ezen belül kell lennie az újonnan felvett VST pluginnek. A hozzáadás után a **2.2**-es kép alapján kell összekötni a Midi Input és Midi Outputtal. A VST blokkra dupla kattintással tudjuk felhozni a saját pluginünk ablakát, ott tudjuk módosítani az értékeket.

2.2 FM szintetizátor felhasználói felülete



2.3 Az FM szintetizátor VST felhasználói felülete.

A Projucer segítségével létrehoztam egy egyszerű VST plugin alkalmazást, amely kiírja, hogy „Hello world.” Az alkalmazáshoz tartozik egy **PluginEditor**. Illetve egy **PluginProcessor** cpp és h fájl. A PluginEditorban kell dolgozni főleg a GUI, azaz a VST felhasználói interfészének megjelenítésével.

Itt van lehetőség összekapcsolni az egyes csúszkákat a konkrét, módosítandó értékekkel. Például, minden egyes ADSR értékhez külön csúszkát rendeltem, ezekért a **slider** osztály felelős, amelynek megjelenése a **2.3**-es képen látható. Egy slidernek több megjelenítési módot be lehet állítani a **setSliderStyle** függvénnyel. A legtöbb csúszkának a „**LinearVertical**” stílust választottam, egyedül a hullámforma választásához rendelt slider objektum kapta a „**RotaryHorizontalVerticalDrag**” formát. A stílus nevében a „HorizontalVerticalDrag” jelenti azt, hogy az egeret milyen irányba húzva lehet megváltoztatni a slider értékét. A horizontális és vertikális slidereknek az értéke tehát felfelé és jobbra mozgatva nő, balra és lefelé pedig csökken. A „RotaryVerticalDrag” beállítással például csak a felfelé és lefelé mozdítással lehetne változtatni az értéken.

A sliderhez egy úgynevezett **label**-t is hozzá lehet rendelni, illetve ezeket slidertől függetlenül is meg tudjuk jeleníteni a felületen. A label-ök szövegdobozok, amelyekben valamilyen írást lehet elhelyezni, ezeket az **attachToComponent**

függvénnyel akár más komponensekhez is hozzá lehet csatolni, például egy sliderhez. Ilyenkor a label pozíciója a sliderétől fog függeni, a **2.3**-as ábrán látható, hogy minden slideremhez tartozik egy label.

Az egyes slider objektumokat a programban használható értékekkel a **AudioProcessorValueTreeState::SliderAttachment** osztállyal lehetséges összekötni. Minden értéknek egy egyedi kulcs kell (például oszcillátor 1 esetén 1GAIN, 1ATTACK stb.), hogy az értéket egyértelműen azonosítani lehessen. Az egyes SliderAttachmentek létrehozásakor meg kell adni ezt az egyedi kulcsot, a slider objektumot, illetve egy AudioProcessorValueTreeState objektumot, ami tárolja az értékeket. Ezeket a tárolókat már a PluginProcessorban kell létrehozni.

2.3 FM szintetizátor belső működése

Maga a hangfeldolgozás főleg a PluginProcessor-ban történik. Itt hoztam létre az oszcillátorok paramétereit tároló AudioProcessorValueTreeState-eket, egyet-egyet az oszcillátoroknak. Ezeket a cpp fájlban inicializálni kell **AudioParameterFloat**-okkal, amiknek meg kell adni ugyan azt az eredeti kulcsot, amit a PluginEditorban is megadtunk. Ezen kívül egy általános leírást is adni kell, aminek nem kell feltétlenül eredetinek lennie (például Attack, Decay, stb.). Specifikálni kell az érték minimumát és maximumát, illetve, hogy mi a kezdeti értéke. A **processBlock** függvény figyeli folyamatosan ezeket az értékeket, és értesíti a szintetizátort a változásokról.

A PluginProcessorban hoztam létre a **Synthesiser** objektumomat, ez egy olyan osztály, amely rendelkezik legalább egy **SynthSound** és **SynthVoice** objektummal. A SynthSound egy passzív osztály, ami egyszerűen leír egy hangot, amit a szintetizátor le tud játaszani. Maga az audió renderelés a SynthVoice-ban történik, ez lehetővé teszi, hogy több SynthVoice objektum egyszerre ugyan azt a hangot le tudja játszani. Ahány SynthVoice objektuma van a szintetizátornak, annyi hangot tud játszani polifonikusan.

A hang kezelésének törzse a SynthVoice-ban rejlik. Itt a **renderNextBlock** függvény a legérdekesebb, amiben egy for ciklus van a mintavételi frekvenciára, ezen belül még egy a csatornák számára (esetemben kettő, mivel sztereóban dolgoztam). Az audió renderelése a következő algoritmus alapján működik:

```
auto modulator = getModulatorFreq();
double sound = osc1Gain * osc1ADSR.getNextSample() * getCarrierFreq(modulator) +
               osc2Gain * osc2ADSR.getNextSample() * modulator;
outputBuffer.addSample(channel, startSample, sound);
```

A **modulator** jelenti a moduláló oszcillátor frekvenciáját, ezt a **getModulatorFreq** függvénnyel kérem le. Ebben a függvényben van egy switch az **osc2Shape** változóra, ami a következő értékeket kaphatja:

0: Szinuszjel

1: Fűrészfogjel

2: Pulzusjel

3: Háromszögjel

Tehát ha ez az érték nulla, akkor egy szinusz jelet kell generálni, ami a **Maximilian maxiOsc** osztály elemére meghívott **sinewave** függvénnyel egyszerűen kivitelezhető. Ennek egyetlen paramétert kell megadni, a hullám frekvenciáját, amelyet a **realFreq** függvénnyel számoltam ki. Itt a játszott frekvenciához hozzáadom a játszott frekvencia és a frekvencia arány szorzatát, ez az arány -0.9999 és 1.0 közötti érték lehet.

Miután megvan a modulátor frekvenciája, kiszámolható a vivőhullám a **getCarrierFreq** függvényben, ahol szintén először egy switch ellenőrzi, hogy milyen hullámformát kell generálni. Itt egy sinusz generálásakor a játszott frekvenciához hozzáadom a modulátor frekvenciájának és a modulációs indexnek a szorzatát, ez a frekvenciamodulálás függvénye. Az index érték határozza meg a frekvencia modulálás mélységét, a frekvenciaeltérés nagyságát, azaz, hogy mennyire fog eltérni a modulált frekvencia az eredetileg játszottól. A kapott hullámot megszoroztam az oszcillátorhoz tartozó **gain** értékkel, illetve az JUCE ADSR osztály segítségével az ADSR értékkel, amit a **getNextSample** fügvénnyel lehet kinyerni. A vivőjel kiszámolása után még a modulátor jelet is hozzá adtam a vivőhöz, ugyanis, ha feltekerem annak a hangerejét, akkor azt is hallani akarom. Mivel a két hullámot külön ADSR és gain értékekkel szoroztam meg, ezért ezek a paraméterek oszcillátoronként külön-külön módosíthatók. A végső hullámforma kiszámolása után hozzáadtam azt a kimeneti bufferhez.

3 Összefoglaló

Összességében egy nagyon érdekes projekt és kihívás volt a szintetizátor elkészítése. Szeretnék a jövőben is foglalkozni az audió programozással, ugyanis a zenei tevékenységek óriási szerepet töltenek be az életemben, és nagyon érdekes az eszközöket, amiket nap mint nap használok programozói szemmel is megtekinteni. Szeretném tovább fejleszteni a szintetizátort különböző modulációs algoritmusokkal, szebben hangzó oszcillátorokkal, filterrel. A kódom és a felhasznált Maximilian könyvtár fájlok elérhetők a Github oldalamon [4].

Ha valakinek szintén felkelti az érdeklődését a téma, nagyon ajánlom a JUCE környezetet. Érthető, jól felépített osztályokat és függvényeket tartalmaz, kiváló a dokumentációja, és óriási, aktív rajongó bázissal rendelkezik. Youtube-on a „The Audio Programmer” csatornát is csak ajánlani tudom, a JUCE-ról készített videói rengeteget segítettek a fejlesztőkörnyezet megértésében [3].

Felhasznált irodalom

[1] Various contributors: *Maximilian, C++ Audio and Music DSP Library* https://github.com/micknoise/Maximilian

[2] JUCE: *Main page* https://juce.com/

[3] Youtube: *The Audio Programmer* https://www.youtube.com/c/TheAudioProgrammer

[4] Lengyel Dániel: *FM synth* https://github.com/Len-Daniel/FMsynth/tree/main