Eötvös Loránd Tudományegyetem

Informatikai Kar

Programozási nyelvek és Fordítóprogramok Tanszék

VST szintetizátor

Témavezető:
Dr. Biró Csaba
Tudományos munkatárs

Szerző:
Habzda Bálint Máté
programtervező informatikus BSc

Budapest, 2023

Tartalomjegyzék

1. fejezet

Bevezetés

1.1. Témaválasztás

Szakdolgozatom egy additív szintetizátor plugin, melyet bármelyik VST3 szabványt támogató hoszt képes futtatni. Ide tartozik az összes szabvány digitális audio munkaállomás (DAW) program is. Az elterjedt hullámformákra (Fűrész, Négyzet, stb.) előbeállítások is találhatók a választható lehetőségek között. Megtalálható minden megszokott beállítás: A szintetizátor hangolható (több, különböző mértékben), a szintetizált jel fázisa módosítható, használható beépített uniszónó hanghatás, amplitúdó burkológörbe, valamint erősítés. Emellett egy effekt-lánc, választható és konfigurálható hanghatásokkal: Ekvalizer, Szűrő, Kompresszor, Delay, Reverb, Kórus, Phaser, Tremoló. A szintetizátor továbbá képes állapotát menteni és betölteni (ez a hoszton keresztül történik).

Már rég óta foglalkoztat ez az irány, de idáig kevés programozási ismeretem volt vele kapcsolatban, ezért jó tanulási lehetőségnek találtam. A szakdolgozat elkészítése közben sokat tanultam a szoftveres hangtechnikai eszközök részletes belső működéséről, a felmerülő konkurrencia problémákról, valamint a plugin ökoszisztéma működéséről.

1.2. Implementáció

A szintetizátor a JUCE keretrendszerre épült, objektum-orientált C++-ban a C++20 nyelvi szabvány szerint. A keretrendszernek köszönhetően az alkalmazás

bármilyen platformon, valamint az összes főbb plugin formátumban (VST2, AAX, AU) vagy akár önálló futtatható programként is építhető.

2. fejezet

Felhasználói dokumentáció

2.1. Rendszerkövetelmények

A program futtatásához és tárolásához meglehetősen kevés erőforrás szükséges. Maga a plugin kb. 6 MB helyet foglal a háttértáron, Memóriában ez 40 MB alatt van. Processzorigénye nem magas, stabil 1 GHz mellett, minden funkcióját kihasználva is képes buffer-időn belül dolgozni, akadozásmentesen.
A plugin futtatásához javasolt követelmények:

- Hardver:
  - Processzor: 2 GHz
  - Memória: 4 GB
  - Háttértár: 32 GB
  - Audio kimeneti eszköz (ajánlott)
  - MIDI-kontroller (ajánlott)
- Szoftver:
  - A program futtatásához szükséges hoszt program, ami támogatja a VST3 bővítmények futtatását. Leggyakrabban ilyen célra az úgynevezett digitális audio munkaállomás (DAW) programok használatosak. pl.: FL Studio, Ableton, Pro Tools, Cubase, Logic Pro (macOS), stb.
  - Mivel a program egy cross-platform keretrendszerben készült, bármilyen operációs rendszeren működőképes, amit a keretrendszer támogat: Windows, macOS, Linux.

  - Windows operációs rendszeren a program futtatásához szükséges a Microsoft Visual C++ Runtime csomag.

Figyelembe véve, hogy a leggyakrabban használt hosztok, valamint operációs rendszerek hardveres rendszerkövetelményei általában sokkal magasabbak a fentieknél, a felhasználónak elég az általa használt hoszt rendszerkövetelményeit figyelembe vennie.

A program skálázható (DPI-aware) grafikus felülettel rendelkezik, így kisebb (pl. notebook) monitorokon is megfelelő méretben jelenik meg. Ajánlott 1920x1080 felbontású kijelző használata.

2.2. Telepítés

2.2.1. Függőségek telepítése

Windows gépeken szükséges a Visual C++ Redistributable[1] könyvtár.

2.2.2. Építés

A VST bővítmények belső formátuma függ a célzott operációs rendszertől.[2] Emiatt a plugint az adott operációs rendszerben kell fordítani. Erre lehetőséget ad a JUCE keretrendszer Projucer[3] alkalmazása. A VST_Synth.jucer fájl egy Projucer projekt, amiben 3 exportáló létezik:

1. Windows: Visual Studio 2022[4]
2. macOS: Xcode[5]
3. Linux: Makefile

A Projucer automatikusan létrehozott fordítási célpontjait használva a fordítás eredménye egy az adott platformhoz tartozó VST_Synth.vst3 fájl, ami az adott operációs rendszeren használható. Továbbá építhető standalone futtatható alkalmazás

[1] https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170
[2] https://steinbergmedia.github.io/vst3_doc/vstsdk/index.html#vstPlugGeneral
[3] https://juce.com/download/
[4] https://visualstudio.microsoft.com/vs/
[5] https://developer.apple.com/xcode/

is, amiben a program átveszi a hoszt szerepét. A fordításhoz csak a JUCE keretrendszer kódja kell, ami tartalmazza a VST3 formátumhoz tartozó kódot is.

2.2.3. Plugin telepítése és futtatása

Az bővítmény telepítése egyszerű. Nem szükséges elő-konfiguráció a plugin részéről. A VST_Synth.vst3 fájlt operációs rendszertől függően a következő mappákba érdemes helyezni:

Operációs rendszer	Elérési útvonal
Windows	C:\Program Files\Common Files\VST3
macOS	Macintosh HD/Library/Audio/plugins/VST3
Linux	/usr/lib/vst3/

2.1. táblázat. Alapértelmezett VST3 mappák különböző operációs rendszereken

A hosztok ezekben a mappákban keresik elsőként a felhasználó bővítményeit. A legtöbb hoszt megengedi, hogy a felhasználó máshol tárolja ezeket, ebben az esetben, a VST_Synth.vst3 fájl bárhol tárolható, amíg a hosztban konfigurálva van az új elérési útvonal.

A fájl megfelelő mappába helyezése, majd a hoszt elindítása után, érdemes indítani egy plugin szkennelést, ha a hoszt nem tette meg automatikusan. Így a hoszt plugin nyilvántartásába kerül a program. Ezután a bővítmény könnyen behelyezhető egy (vagy akár több) hangszer-csatornába.

2.3. Felhasználói felület

A bővítmény betöltése után, azt megnyitva a szintetizátor grafikus felülete fogad minket. Ez a felület átméretezhető, az alapértelmezett méretnél nagyobb méretekre is. A szintetizátor összes paramétere módosítható egérrel, valamint az értékek billentyűzettel is átírhatók. A paraméterek nagyrésze a hoszton keresztül is módosítható és automatizálható.
A felület 2 fő elemből áll:

1. Szerkesztő
2. Billentyűzet

2.3.1. Szerkesztő

A szerkesztőben lehet módosítani a szintetizátor paramétereit. A szerkesztő felülete 3 fülből áll:

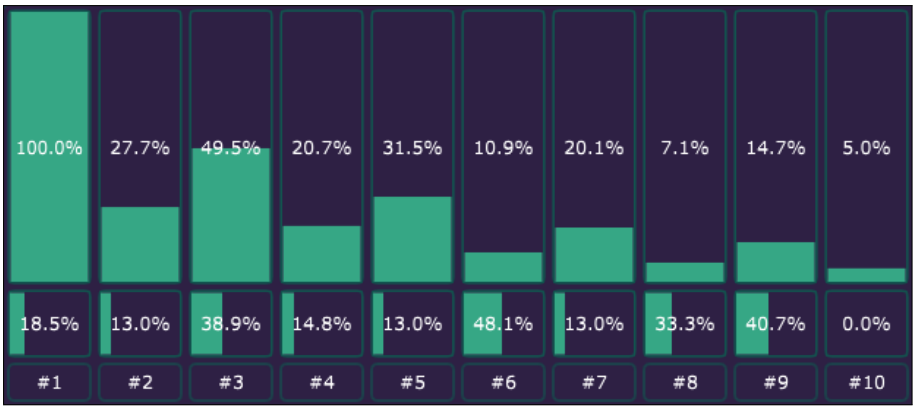
1. Oscillator
2. Synthesizer (alapértelmezett)
3. Effects

Oscillator

Ezen a fülön található a szintetizált hullám képe, valamint az oszcillátor összes paraméte. Ezek a paraméterek a hullám egy-egy harmonikusának relatív hangerejét és fázisát reprezentálják.

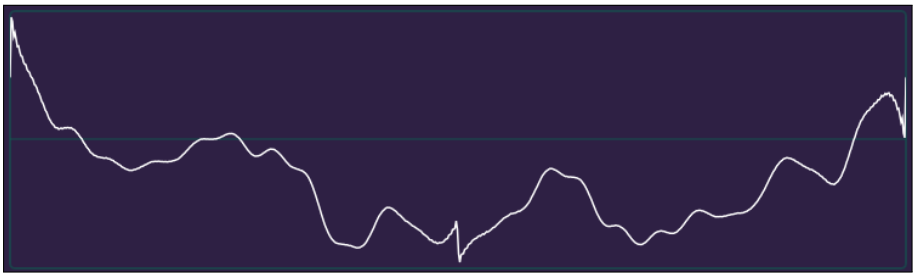
2.1. ábra. Oscillator fül megjelenése

Az oldal alsó felében a harmonikusok paraméterei vannak felsorolva. Mind a 256 harmonikushoz egy hangerő- és egy fázis csúszka, valamint egy a harmonikus sorszámát tartalmazó címke tartozik. Ez a felület görgethető. #1 az alaphang.



2.2. ábra. Az első 10 harmonikus paraméterei

Az oldal felső felében egy oszcilloszkóp található, ami a szintetizált hullám egy teljes fázisát rajzolja a képernyőre. Az oszcilloszkóp valós időben jelzi a hullámon végzett módosításokat.



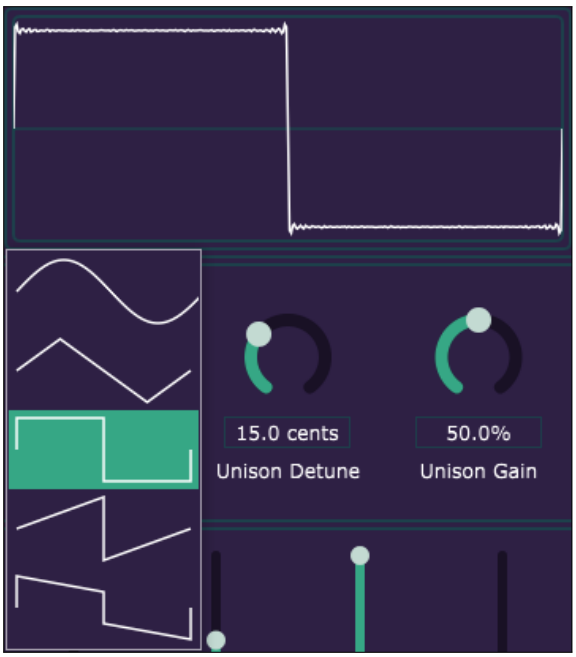
2.3. ábra. Oszcilloszkóp

Synthesizer

A Synthesizer fülön található a szintetizátor összes fő paramétere.

2.4. ábra. Synthesizer fül megjelenése

A bal felső sarokban egy oszcilloszkóp van. Rákattintva előjön egy menü, ami előre beállított hullámokat tartalmaz. Betöltés után, ezek a hullámok módosíthatók az Oscillator fülön keresztül.



2.5. ábra. Hullám választó

Bal középen találhatók az uniszónó effektus paraméterei:

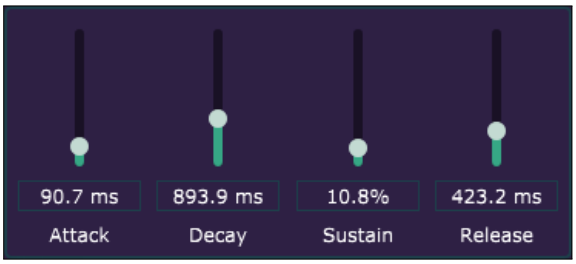
- A `Unison Pair Count` paraméter határozza meg, hogy hány pár extra hullámot generál a szintetizátor, az alaphang mellett.

- A `Unison Detune` paraméter határozza meg, hogy az első uniszónó pár hány centtel van elhangolva az alaphangtól. A pár egyik hangja fel-, a másik le van hangolva. A többi pár egyenletes eloszlással van hangolva a `Detune` paraméter és az alaphang között.
- A `Unison Gain` paraméter határozza meg az uniszónó hang hangerejét. 100%-nál minden uniszónó hang egyezik az alaphang amplitúdójával.

2.6. ábra. Uniszónó paraméterek

Bal alul található a szintetizátor által generált hangok burkológörbéinek paraméterei:

- Az `Attack` paraméter határozza meg a hang felfutási idejét
- A `Decay` paraméter határozza meg a hang visszaesési idejét a kitartási szintre. Ez a fázis addig tart, amíg a hangjegy tart.
- A `Sustain` paraméter határozza meg a hang kitartási amplitúdóját. Amíg a hangjegy tart, ezen a szinten fog maradni a hullám.
- A `Release` paraméter határozza meg a hang lecsengési idejét, miután a hangjegy véget ért.



2.7. ábra. Burkológörbe paraméterek

A jobb felső sarokban található a szintetizátor hangjainak fázis paraméterei:

- A `Phase` paraméter az összes generált hullám (uniszónót beleértve) fázisát egyszerre módosítja.
- A `Phase Randomness` paraméter minden egyes generált hullám fázisát véletlenszerűen módosítja 0 és a kiválasztott százalék között. Ez a paraméter, a módosítás után kezdődő hangjegyeken változtat.

2.8. ábra. Fázis paraméterek

Jobb középen találhatók a szintetizátor hangolási paraméterei:

- Az `Octaves` paraméter a szintetizátort teljes oktávokkal hangolja
- A `Semitones` paraméter a szintetizátort félhangokkal hangolja
- A `Cents` paraméter a szintetizátort centekkel hangolja
- A `PW Range` paraméter a MIDI Pitch Wheel hajlítási határait adja meg félhangokban

2.9. ábra. Hangolási paraméterek

A jobb alsó sarokban található a szintetizátor hangerőszabályzója és egy sztereó RMS szintjeltő.

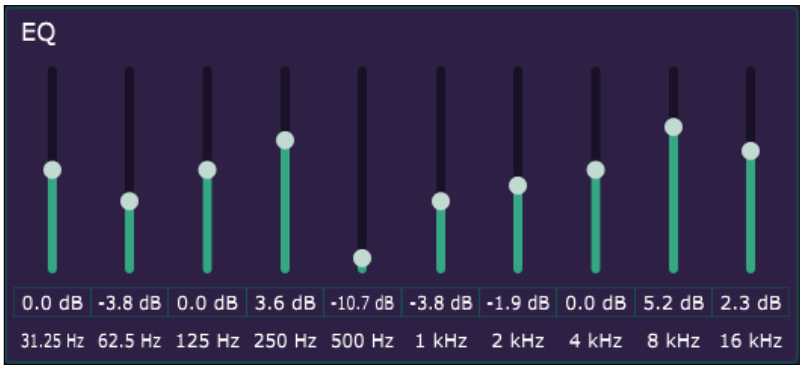
2.10. ábra. Hangerő szabályzó és szintjelző

Effects

Ezen a fülön találhatók az Hangeffekt-lánchoz tartozó paraméterek. Bal oldalon a lánc egységeinek választói találhatók, jobb oldalon a kiválasztott effektek paramétereit tartalmazó komponensek jelennek meg a láncon foglalt helyüknek megfelelő sorrendben.

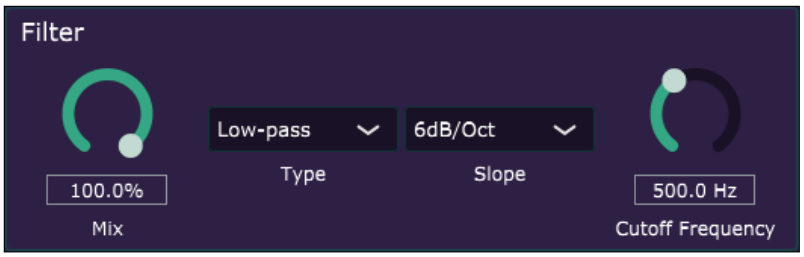
2.11. ábra. Effects fül megjelenése

EQ Az EQ effekt egy proportional Q ekvalizer. Minden sáv sávszélessége az erősítés nagyságától függ. Alacsony erősítésnél széles, magasnál keskeny. Minden sáv erősítése ±12 dB között állítható.



2.12. ábra. EQ komponens

Filter A Filter effekt egy szűrő.



2.13. ábra. Filter komponens

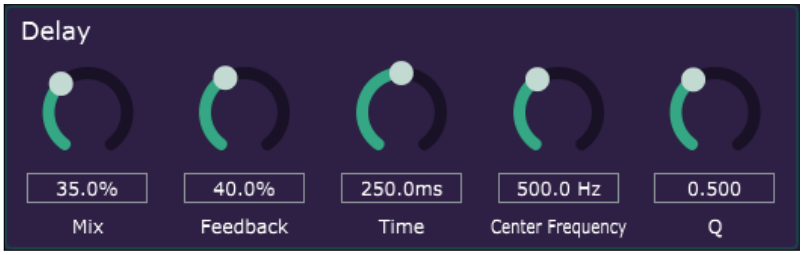
- A `Mix` paraméter határozza meg a szűrő kimenetének arányát az eredeti jelhez képest.
- A `Type` választó a szűrő típusát határozza meg. Lehet aluláteresztő (Low-pass) vagy felüláteresztő (High-pass).
- A `Slope` paraméter a szűrő meredekségét határozza meg dB/oktávban.
- A `Cutoff Frequency` paraméter a szűrő vágási frekvenciáját határozza meg.

Compressor A Compressor effekt egy kompresszor.

2.14. ábra. Compressor komponens

- A `Mix` paraméter határozza meg a kompresszor kimenetének arányát az eredeti jelhez képest.
- A `Threshold` paraméter kompresszor küszöb paramétere. Ha a jel hangereje a `Threshold` paraméter felé emelkedik, akkor a `Threshold` felé emelkedő szint csökken.
- A `Ratio` paraméter a szintetizátor szűkítési arányát határozza meg. A `Threshold` feletti hangerő ennek arányában csökken. Ha 1:1, a jel nem változik.
- Az `Attack` paraméter a kompresszor felfutási idejét határozza meg. Miután a jel eléri a `Threshold` szintet, ennyi időbe telik, mire a kompresszor `Ratio`val csökkenti a `Threshold` feletti szintet.
- A `Release` paraméter a kompresszor lecsengési idejét határozza meg. Miután a jel a `Threshold` szint alá esik, ennyi időbe telik, mire a kompresszor abbahagyja a szűkítést.

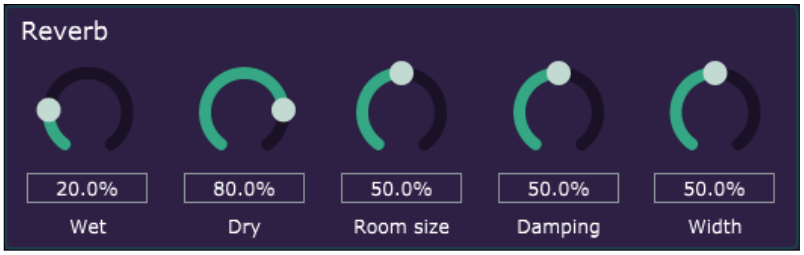
Delay A Delay egy késleltetés hangeffekt szűrt visszacsatolt jellel.



2.15. ábra. Delay komponens

- A `Mix` paraméter határozza meg a késleltetés kimenetének arányát az eredeti jelhez képest.
- A `Feedback` paraméter a visszacsatolás arányát határozza meg.
- A `Time` paraméter a késleltetés hosszát határozza meg.
- A `Center Frequency` paraméter a visszacsatolt jel sáváteresztő szűrőjének középfrekvenciáját határozza meg.
- A `Q` paraméter a visszacsatolt jel sáváteresztő szűrőjének sávszélességét határozza meg (a középfrekvenciával fordított arányban).

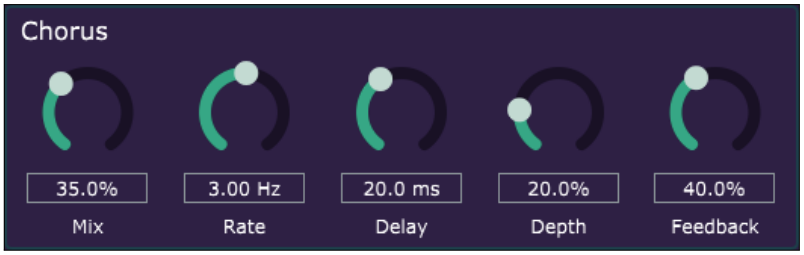
Reverb A Reverb egy algoritmikus visszhang effekt.



2.16. ábra. Reverb komponens

- A `Wet` paraméter határozza meg a visszhang kimenetének hangerejét.
- A `Dry` paraméter határozza meg az eredeti jel hangerejét.
- A `Room Size` paraméter a visszhang hosszát határozza meg.
- A `Damping` paraméter a visszhang tompítását határozza meg.
- A `Width` paraméter a visszhang sztereó szélességét határozza meg.

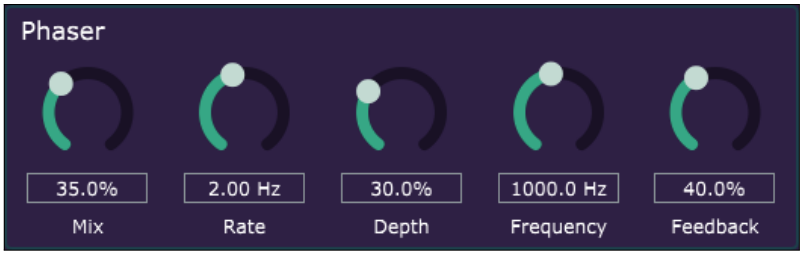
Chorus A Chorus egy kórus effekt.



2.17. ábra. Chorus komponens

- A `Mix` paraméter határozza meg a kórus kimenetének arányát az eredeti jelhez képest.
- A `Rate` paraméter a kórus modulációs sebességét szabályozza.
- A `Delay` paraméter a kórus késleltetését határozza meg.
- A `Depth` paraméter a kórus modulációs mélységét határozza meg.
- A `Feedback` a kórus visszacsatolási paramétere.

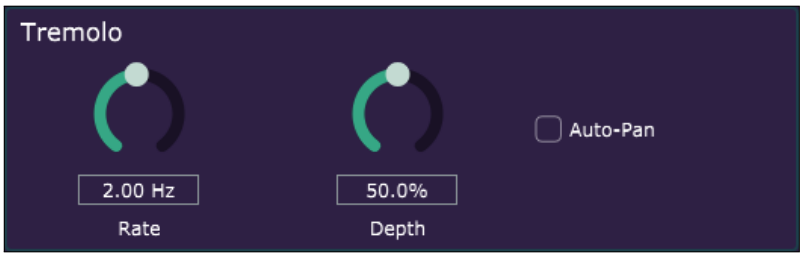
Phaser A Phaser egy fázis-moduláló effekt.



2.18. ábra. Phaser komponens

- A `Mix` paraméter határozza meg a phaser kimenetének arányát az eredeti jelhez képest.
- A `Rate` paraméter a phaser modulációs sebességét szabályozza.
- A `Depth` paraméter a phaser modulációs mélységét határozza meg.
- A `Frequency` paraméter a phaser középfrekvenciáját határozza meg.
- A `Feedback` a phaser visszacsatolási paramétere.

Tremolo A Tremolo a tremoló hatást szimulálja. A jel amplitúdóját modulálja.



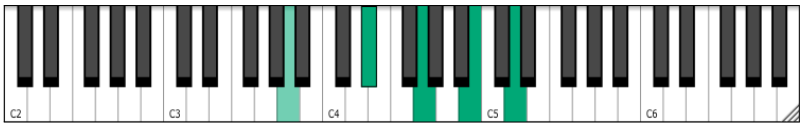
2.19. ábra. Tremolo komponens

- A `Depth` paraméter a tremoló modulációs mélységét határozza meg.
- A `Rate` paraméter a tremoló modulációs sebességét szabályozza.
- Az `Auto-Pan` kapcsoló határozza meg, hogy a moduláció sztereóban (be) vagy monóban (ki) történik.

2.3.2. Billentyűzet

A billentyűzet használható egérrel, valamint szöveges billentyűzettel is. A leütött billentyűkre úgy válaszol a szintetizátor, mint a hoszttól származó MIDI üzenetekre.

A billentyűn látható az összes jelenleg leütött billentyű, a hoszttól érkező hangjegyekkel együtt. Alapértelmezett méreten a 2-6 oktávok (ahol a középső C = C4, alapértelmezetten 261.63 Hz) láthatók rajta, de ha a szintetizátort nagyobbra méretezzük, megjelenik az első oktáv is.



2.20. ábra. Billentyűzet

2.4. Általános használat

A pluign lejátszás közben is reagál a paraméterek változására így az használható valós idejű audio szerkesztésre, élő előadásra, renderelésre.

A paraméterek nem csak a plugin grafikus felületén, hanem a hoszton keresztül is módosíthatók, valamint a hoszthoz csatolt MIDI interfészek szabályzói is köthetők a paraméterekhez. Zeneszerkesztés és renderelés esetén ezek a paraméterek automatizálhatók, így időben is megszabható a szintetizátor hangzása. Élő játék közben a MIDI interfész szabályzóival érhető el hasonló hatás. A plugin grafikus felülete a külső módosításokhoz is igazodik.

A hosztok támogatják a pluginek állapotának mentését. Egy projekt mentése és megnyitása után is elérhető marad minden korábbi módosítás. A legtöbb hosz támogatja, hogy a plugin állapotát projekten kívül is elmentsük, így több konfiguráció előre beállítható és bármikor cserélhető.

3. fejezet

Fejlesztői dokumentáció

A feladat pontos leírása és megoldása több kihívást is jelentett:

- A program szükséges funkcióinak és követelményeinek meghatározása, felületének megtervezése
- A VST3 specifikáció megismerése és ennek összekötése a program funkcióival
- A nyelv és keretrendszer kiválasztása és megismerése
- A program tervezése, figyelembe véve a keretrendszer képességeit és korlátait
- A program implementálása

3.1. Specifikáció

3.1.1. Nem funkcionális követelmények

Termék követelmények

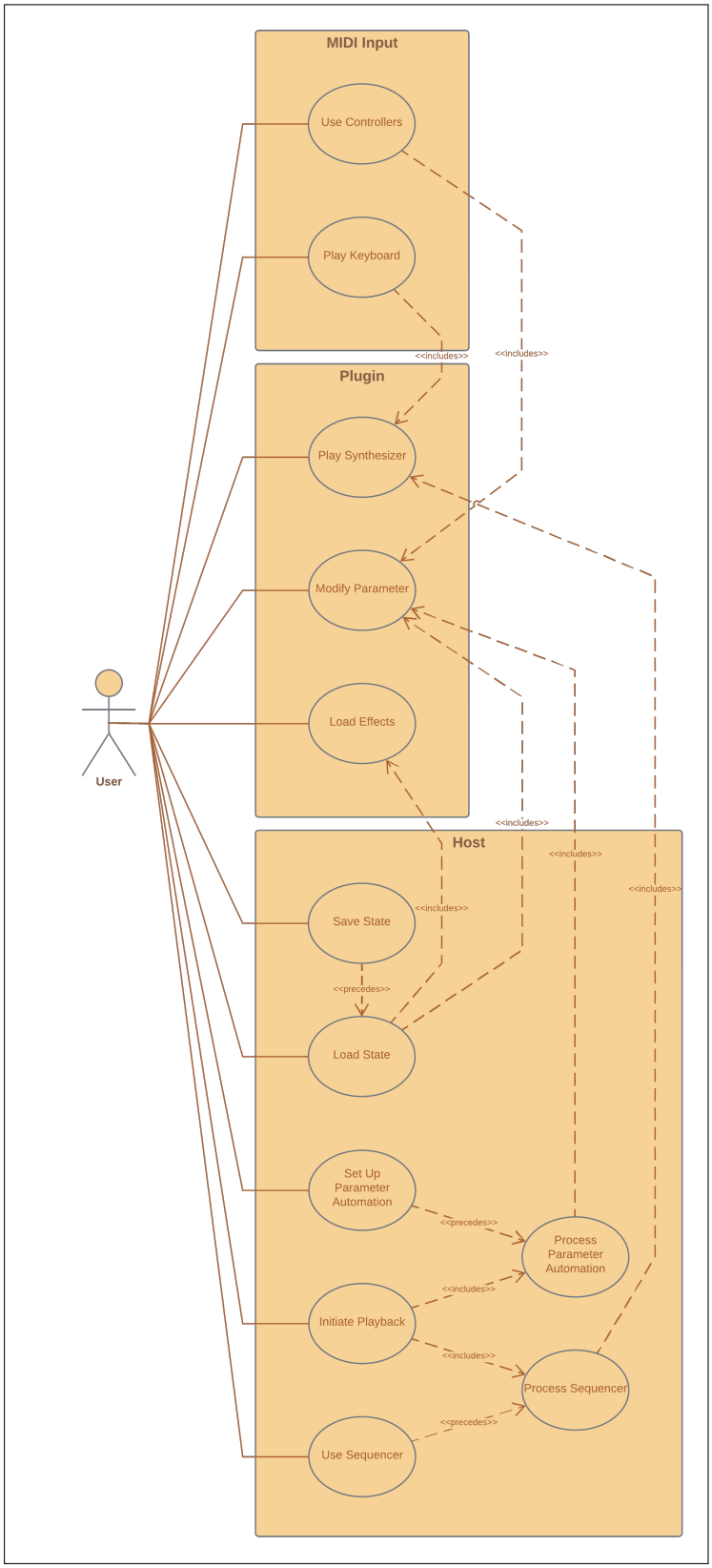
- Hatékonyság:
  - Teljesítmény: A program képes hiba nélkül reagálni a bemenetekre minden 10 ms-nál nagyobb buffer méret mellett 48000 Hz mintavételi frekvenciánál.
  - Mérte: A program nem foglal sok helyet.
- Megbízhatóság: Minden funkcionalitás használható hibák keletkezése nélkül. Az audio kimenetben és a grafikus felületen nincsenek akadások.

- Biztonság: A program nem végez fájl műveleteket, nem fér a rendszerhez, önmaga nem tárol adatokat.
- Hordozhatóság: Minden platformra építhető és felhasználható.
- Felhasználhatóság: Producerek és előadók számára intuitív és ismerős felület.

Menedzselési követelmények

- Környezeti: Hoszt program szükséges a futtatásához, a VST3 specifikációnak meg kell felelnie
- Működési: Bármilyen hosszú használati idő bármilyen gyakorisággal.
- Fejlesztési:
  - C++ nyelv, JUCE keretrendszer
  - Objektumorientált paradigma

3.1.2. Használati esetek



3.1. ábra. Use case diagram

3.1.3. Felhasználói történetek

AS A		User
1.	GIVEN	A program állapota korábban módosítva volt
	WHEN	A hosztban mentett projekt fájl betöltésre kerül
	THEN	A program korábbi állapota visszaáll
2.	GIVEN	A program fut
	WHEN	Leüt egy billentyűt
	THEN	Egy szólam elindul
3.	GIVEN	Van vagy nincs aktív szólam
	WHEN	Megváltoztat valahány paramétert az Oscillator fülön
	THEN	Az aktív szólamok hang kimenete azonnal megváltozik
4.	GIVEN	Van vagy nincs aktív szólam
	WHEN	Megváltoztat valahány paramétert a Synthesizer fülön a fázis és burkológörbe paraméterek kivételével
	THEN	Az aktív szólamok hang kimenete azonnal megváltozik
5.	GIVEN	Van vagy nincs aktív szólam
	WHEN	Megváltoztatja a fázis vagy burkológörbe paramétereket
	THEN	A jövőbeli szólamok hang kimenete megváltozik
5.	GIVEN	Van aktív hang kimenet
	WHEN	Betölt, kitöröl egy effektet vagy módosít egy effekt paramétert
	THEN	A hang kimenet megváltozik a módosításnak megfelelően

3.1. táblázat. User Story-k

3.1.4. Funkciók specifikációja

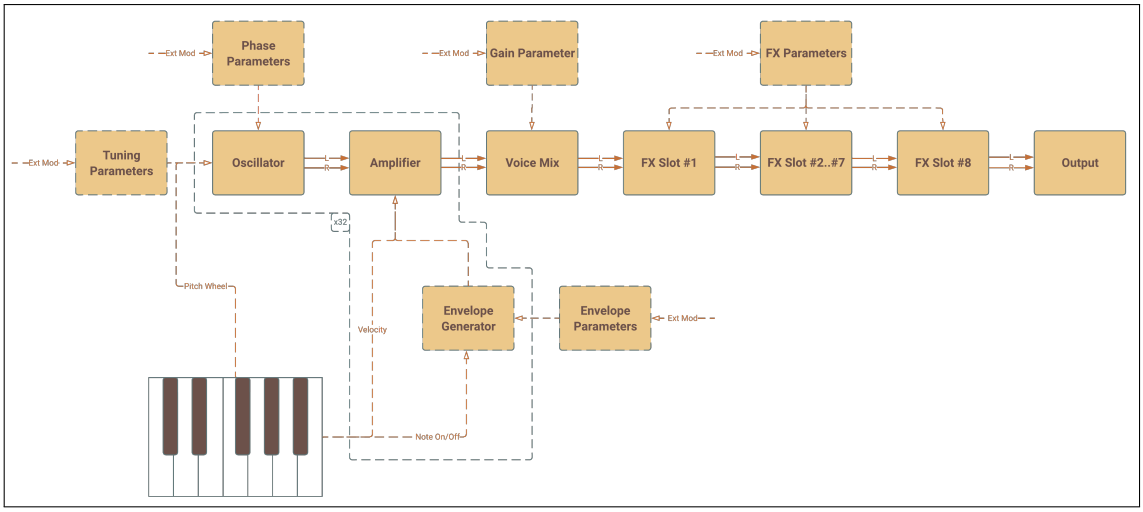
A program funkcióinak meghatározásához szükségem volt korábbi tapasztalataimra hasonló stúdió programok használatáról. Mindenképp szerettem volna egy additív oszcillátort, ami egy önkényesen megadott periodikus hullámot képes létrehozni. Fontos volt továbbá, hogy a szintetizátor sok szólamot tudjon kezelni. Végül 32 mellett döntöttem. Ennél több ritkán szükséges, a legtöbb felhasználói esetben 10-nél kevesebb van használatban.

Egy szintetizátornak mindenképpen szüksége van valamilyen belső hangolási paraméterekre, továbbá lehetőséget kell adnia a MIDI pitch wheel használatára, mivel ezek használatát külső módszerekkel bonyolultabb és aránylag sokkal költségesebb utólag szimulálni.[1] Ezeknek határait a tapasztalataim szerint adtam meg (±2 oktáv, ±12 félhang, ±100 cent, valamint a MIDI pitch wheel maximális hajlítási mértéke ±12 félhangra állítható).

[1] `https://www.guitarpitchshifter.com/algorithm.html`

Szükség van még egy burkológörbére, ami minden szólamban külön változtatja a jel erősítését, valamint az uniszónó hatásra, ami a hangoláshoz hasonlóan nehezen imitálható, miután a szólamok össze vannak keverve és elér a jel az effektekig.

Az effekt láncnál a cél, hogy minden hatás bármilyen sorrendben használható legyen.



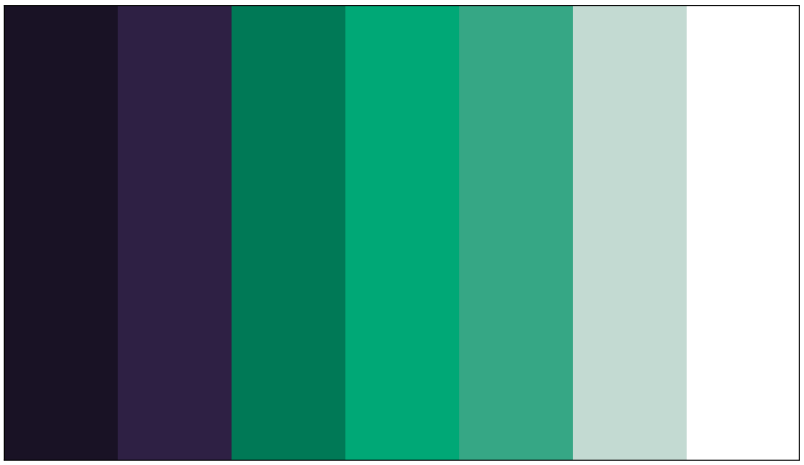
3.2. ábra. Signal Flow Diagram

3.1.5. Felület

A felülettel kapcsolatban két célom van: átlátható és intuitív kell legyen egy hozzáértő szemében.

Átlátható

Fontos, hogy a használt színek mellett mindennek olvashatónak kell maradnia. Továbbá minden különböző célt ellátó komponensnek elkülöníthetőnek kell lennie a többitől. Az effekteknél fontos, hogy csak azok legyenek láthatók, amik használatban vannak, valamint, hogy a felület jelezze, ha egy effekt már be van töltve, és emiatt máshol már nem használható.



3.3. ábra. A felület színskálája

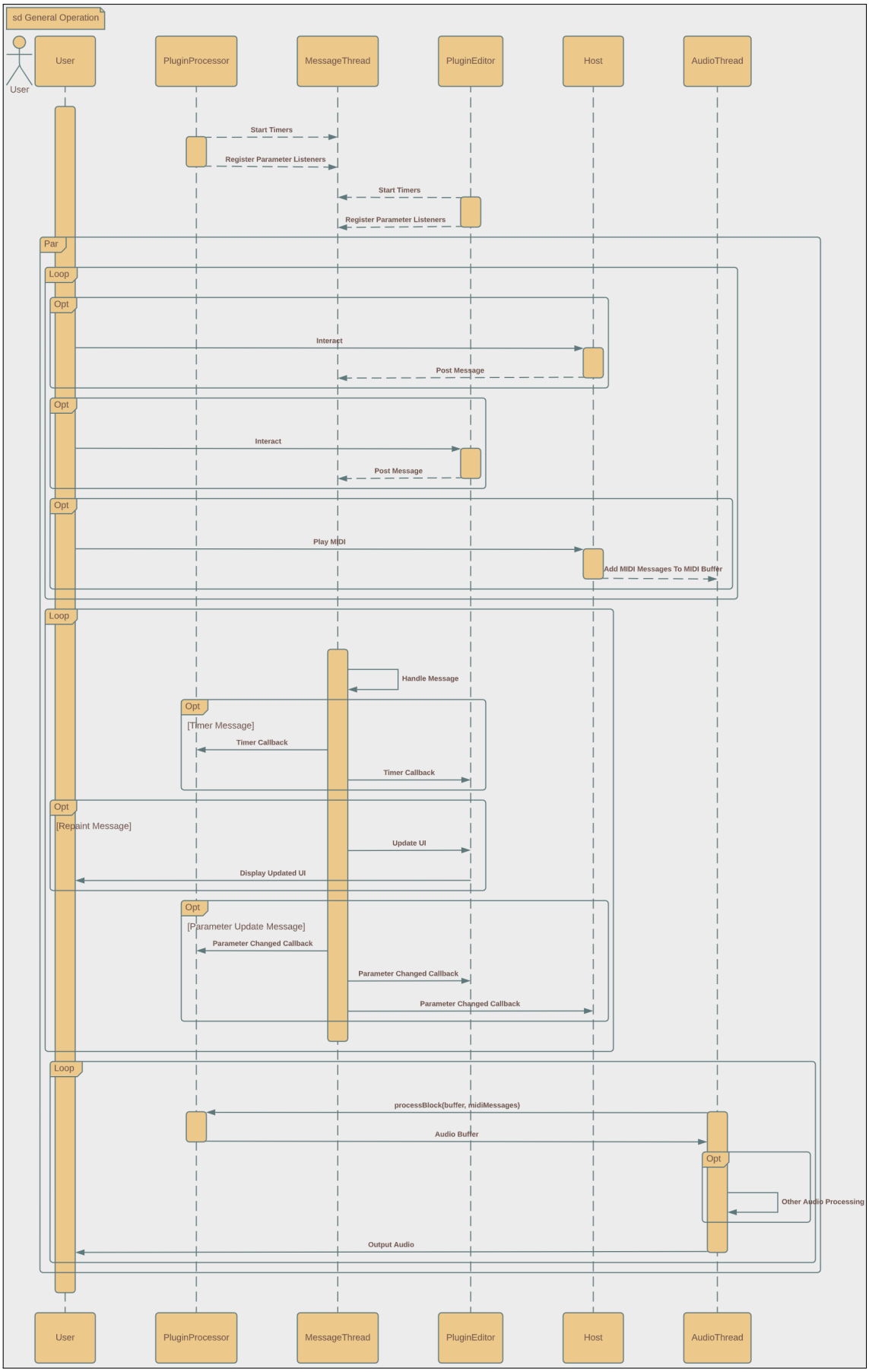
Intuitív

Egy hozzáértő felhasználónak akár a dokumentáció nélkül is fel kell tudni ismernie, hogy melyik paraméter hogyan fog hatni a jelre szimplán a paraméter elhelyezése és neve alapján. Ez az intuíción alapuló folyamat határozza meg a felhasználói tapasztalatot az első használat alatt.

3.2. VST3

A VST3 specifikáció határozza meg, hogy egy VST3 bővítmény hogyan kezelje az bemeneti és kimeneti audio buffereket, MIDI eseményeket, valamint a paraméterekkel kapcsolatos kommunikációt a hoszt és a plugin között.[1]

A legnagyobb megkötés, hogy paramétereket csak a plugin példányosítása során lehet létrehozni, utána nem. Továbbá a hosztok egy plugin minden paraméterét egy-egy azonosító indexxel ismernek fel.



3.4. ábra. A plugin általános működési ciklusa

3.3. JUCE és C++

A JUCE egy objektum-orientált cross-platform keretrendszer főként audio alkalmazások fejlesztésére. A keretrendszer C++ nyelvre épül. Plugin fejlesztésre támogatja az összes főbb plugin formátumot.

3.3.1. C++

A C++ egy jól optimalizálható, valamint kiszámítható választás, idő-kritikus feladatok megoldására. Ilyen esetekben sokat számít a memóriakezelés módja, valamint a beépített adatszerkezetek sebessége.

A legfontosabb szempont a kiszámíthatóság: C++-ban nincs szemétgyűjtés, ami problémákat tud okozni rövid bufferek mellett, ha megállítja az audio szálat.

Sokat számít még, hogy az audio piacon mindenki C++-ban fejlesztett az utóbbi évtizedekben. A témában sokkal több információ és implementáció elérhető ebben a nyelvben, mint bármely másikban.

3.3.2. JUCE

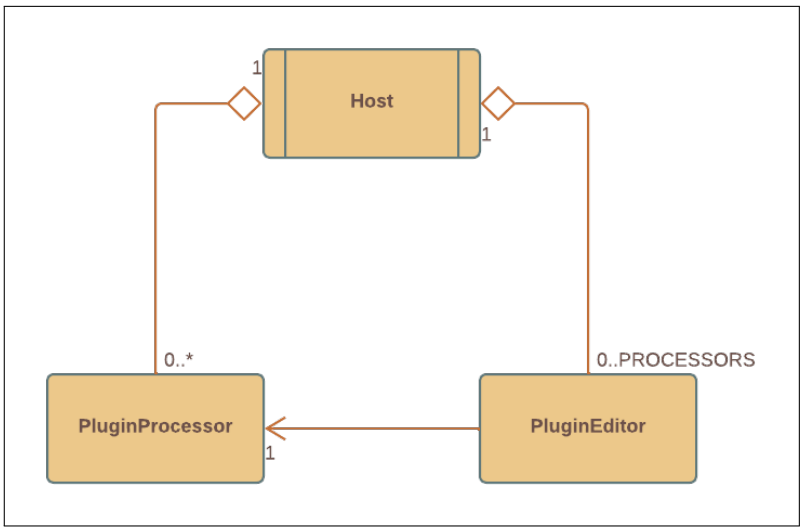
A VST3 specifikáció feltételeinek JUCE VST3 wrapper tesz eleget. Főként emiatt, valamint az integrált grafikus felület fejlesztési lehetősége miatt választottam a JUCE keretrendszert. A VST3 formátum kezelése magába foglalja a processor és controller gyártását, valamint összekötését, és a paraméterekkel és MIDI eseményekkel kapcsolatos kommunikációt is. Így az időm jelentős részét a paraméterek meghatározásával és az audio programozással töltöttem.

A VST3 index-azonosított paraméterei helyett a keretrendszer stringekkel azonosítja a belső paramétereit, a két reprezentáció összekapcsolásáért szintén a VST3 wrapper felelős.

A JUCE keretrendszerben a plugin paramétereinek tárolásáért és az állapotának fenntartásáért az `AudioProcessorValueTreeState` osztály felelős. Ezen osztály egy példányán keresztül elérhető a plugin minden paramétere.
A keretrendszerben a plugin típusú programok két részre vannak bontva:

1. PluginProcessor
2. PluginEditor

Ez a két rész hasonlít egy alkalmazás Model és View egységeire, de a két egység külön áll egymástól, mivel a PluginEditor komponenseinek élettartama nem egyezik a PluginProcessoréval. A kommunikáció a két egység között gyakran a hoszton keresztül történik.

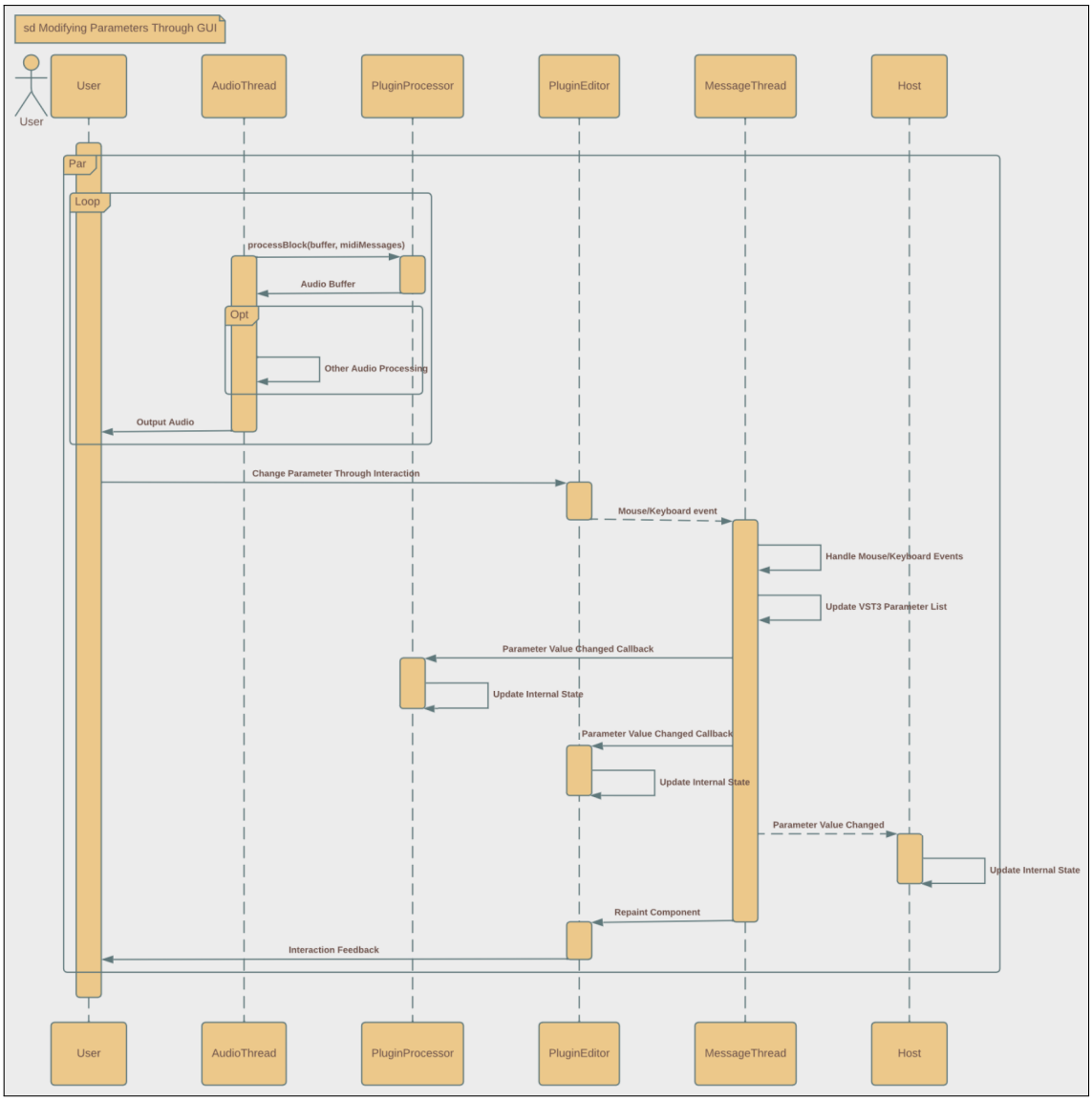


3.5. ábra. Egy JUCE plugin általános felépítése

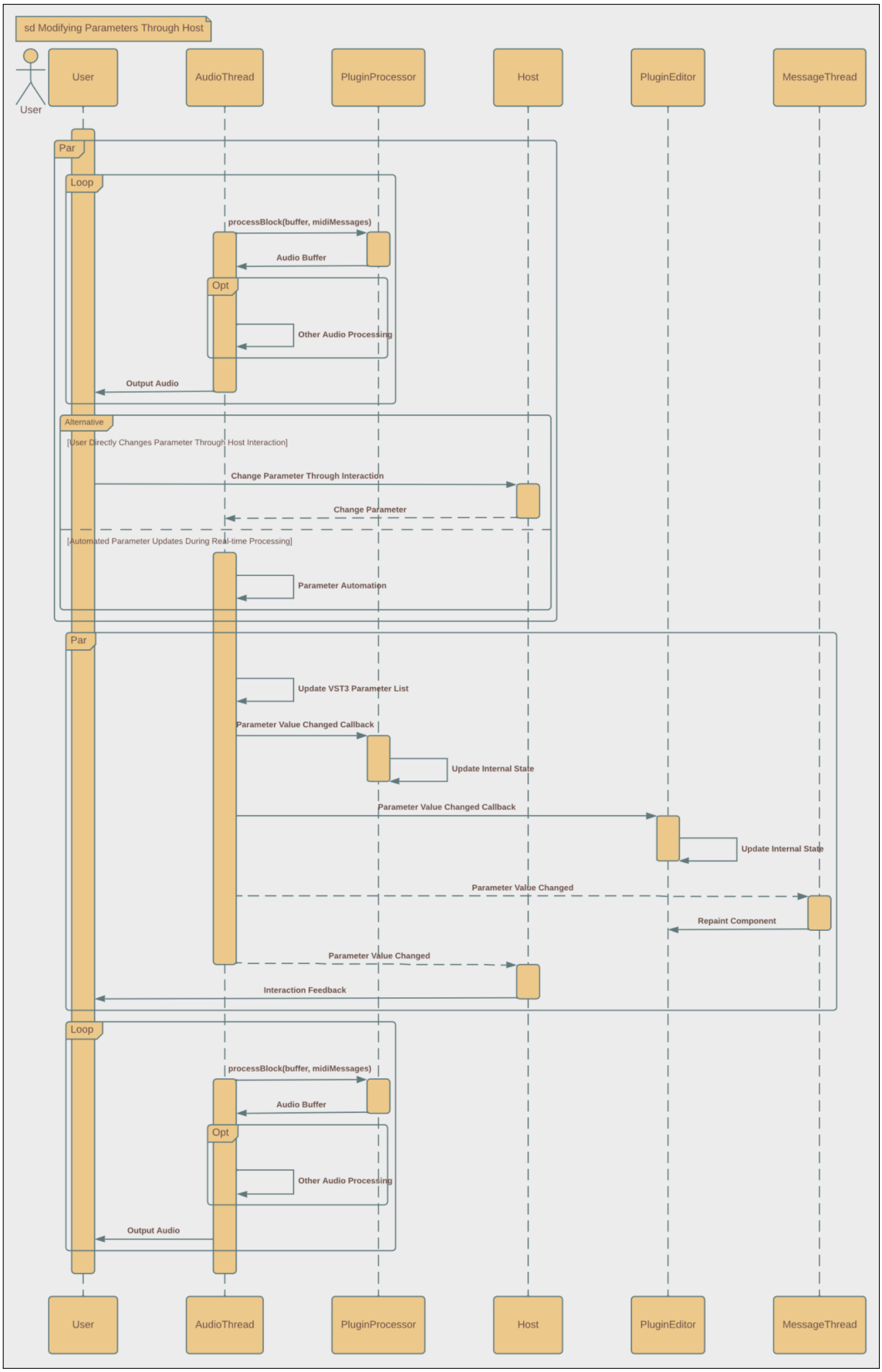
A keretrendszer belső paraméter kezelése eltér a VST3 specifikációban leírt külső kommunikációtól, mivel több plugin formátum specifikációját is támogatja. A belső működés szerint egy paraméter módosításának visszahívó függvényei, valamint a hoszt értesítése szinkron módon történnek, de ennek nincs garantált sorrendje.[2]

Továbbá nem garantált az sem, hogy melyik szálon fog történni a paraméterek változásainak feldolgozása. A hoszt GUI/Message szálain történő paraméter változások (pl. a plugin felületén történő felhasználói interakciók által) azon a szálon lesznek feldolgozva, viszont a hoszton keresztül történő módosításokra nincs ilyen garancia: Egy automatizált paraméter esetén előfordulhat, hogy az audio szálon történnek a paraméterekhez csatolt függvények visszahívásai.[2]

[2]JUCE forráskódja alapján: `https://github.com/juce-framework/JUCE/blob/69795dc8e589a9eb5df251b6dd994859bf7b3fab/modules/juce_audio_processors/processors/juce_AudioProcessor.cpp#L1508`



3.6. ábra. Paramétermódosítás a plugin felületén keresztül



3.7. ábra. Paramétermódosítás a hoszton keresztül

3.4. Programstruktúra és implementáció

A programom 2 fő és 1 mellék névtérből áll:

- Processor
- Editor
- Utils

A 2 fő névtér objektumai írják le a teljes program struktúráját.[3]

A programban a szálak között minden kommunikáció blokkolásmentesen működik, atomikus változók és tripla bufferelés segítségével.

3.4.1. Processor

Ez a névtér tartalmazza a plugin belső állapotát kezelő logikát, valamint a plugin részegységeinek hierarchikus felépítését.

Ennek gyökere a `PluginProcessor.h` és `PluginProcessor.cpp` fájlokban meghatározott `VST_SynthAudioProcessor` osztály. A hoszt ezt az osztályt példányosítja és kezeli a JUCE keretrendszeren keresztül. Amikor a plugin példányosításra kerül létre jön a plugin paramétereit tartalmazó `apvts:AudioProcessorValueTreeState` tag objektum is. A processzor két fő alegysége az `additiveSynth:AdditiveSynthesizer` és az `fxChain:EffectProcessorChain` tag objektumok.

A `juce::AudioProcessor`-ból származó osztályokban három függvény felelős az audio feldolgozásért, a következő sorrendben hívja őket a hoszt:

1. `prepareToPlay`: Átadja a processzornak a feldolgozási kontextus adatait: a mintavételi frekvenciát minták/másodpercben és az audio bufferek maximális méretét minták számában.
2. `processBlock`: Az audio bufferek feldolgozása ebben a függvényben történik. A hoszt vagy az operációs rendszer periodikusan hívja. A feldolgozás ideje nem lépheti túl ezt a periódust. Továbbá a periódus nem konstans: Előfordulhat, hogy a buffer méret változik két hívás között.

[3] A teljes program osztály diagrammja a csatolmányban található

3. `releaseResources`: Ideiglenes adatokat ebben a függvényben lehet felszabadítani. Egy `releaseResources` hívás után a következő `prepareToPlay` hívásig nem történik `processBlock` hívás.

A processBlock hívás a két alegység processBlock-ját hívja egymás után. Először az `additiveSynth:AdditiveSynthesizer`, majd az `fxChain:EffectProcessorChain` dolgozza fel a buffereket. Így a szintetizátor által generált és összekevert jel megy át az effekt láncon.

`apvts:AudioProcessorValueTreeState`

Az `apvts:AudioProcessorValueTreeState` objektum tartalmazza a plugin paramétereit. Mivel a specifikáció nem engedi a paraméterek dinamikus példányosítását, a teljesen dinamikus pluginek gyakran előre lefoglalnak valamennyi generikus paramétert, amiket a felületükön keresztül lehet a belső logikai paraméterekhez csatolni. Az én esetemben elég volt az oszcillátor, a szintetizátor, az effekt lánc és az effekt egységek paramétereit felvenni. Ezt az egységek statikus `createParameterLayout()` függvényei teszik. Emiatt a teljes effekt láncban adott időben minden egységből maximum egy példány létezhet, mivel egy adott effekt egység minden példánya ugyanazokba a paraméter azonosítókra hallgat a belső állapotát illetően.

Továbbá ezzel az objektummal tudja később a hoszt a plugin állapotát menteni és betölteni a `getStateInformation` és `setStateInformation` függvények hívásával.

Mivel ez az egy objektum tárolja a plugin összes paraméterét, a plugin minden egységének ismernie kell, hogy a saját paramétereit el tudják érni.

`Processor::Synthesizer`

A `Processor::Synthesizer` névtér tartalmazza a szintetizátor első fő blokkjának osztályait:

- `AdditiveSynthesizer`
- `OscillatorParameters`
- `AdditiveSynthParameters`
- `AdditiveVoice`
- `AdditiveSound`

`AdditiveSynthesizer:` Ez az osztály a szintetizátor fő egysége. Az `AdditiveSynthesizer` a `juce::AudioProcessor` osztályból származik, így kevés módosítással egy különálló plugin csinálható belőle, vagy akár kicserélhető egy másik szintetizátorra.

Az `AdditiveSynthesizer` tartalmazza az `oscParameters:OscillatorParameters` és a `synthParameters:AdditiveSynthParameters` paraméterleíró tag objektumokat, valamint a `synth:juce::Synthesiser` és a `synthGain:juce::Gain` tag objektumokat.

Az `AdditiveSynthesizer processBlock` függvényében először a `synth:juce::Synthesiser renderNextBlock` hívása történik. Ebben a hívásban a MIDI bemenet és a szólamok kezelése és kimeneteinek összegzése történik. Ezután a szintetizátor alkalmazza a `synthGain:juce::Gain process` függvényét az egész kimenetre.

A `juce::Synthesiser` `juce::SynthesiserVoice`-ból származó `AdditiveVoice` objektumokat kezel, amik a szintetizáror egy-egy szólamát reprezentálják.

A `juce::Gain` objektum egy egyszerű erősítő.

`OscillatorParameters:` Ez az osztály az additív oszcillátor állapotát írja le: Egy periodikus harmonikus hullám első 256 (konfigurálható a `HARMONIC_N` paraméterrel) harmonikusának amplitúdójait és fázisait. A hullámból egy mintát a `getSample` függvénnyel lehet kiszámolni. 256 db harmonikus esetén ez mintánként maximum 256 szinusz érték kiszámítást jelenti, ami 48 kHz mintavételi frekvencia mellett másodpercenként maximum $12'288'000$. Ez csak egy szólam egy hullámának generálásához elég. De a szintetizátoromnak egy időben maximum 32 szólamot kell kezelnie, emellett az uniszónó párok mind 2-2 hullám generálását vonják maguk után, szólamonként (így a szintetizátornak egyszerre összesen maximum $4'325'376'000$ szinusz értéket kéne kiszámolni másodpercenként). Ez meglehetősen sok processzor időbe telne, még a mai modern processzorokon is, ahol specifikus trigonometriai instrukciók vagy dedikált hardver segíti elő ezeket a számításokat. Könnyen belátható, hogy egy gyors, modern processzor architektúra és fordító optimalizálási módszeri mellett is kevés lenne a bufferelésre rendelkezésre álló idő a számítások elvégzésére. Mivel az audio hívás ciklus-iterációinak száma sem előre belátható (pl.: megváltozhat a mintavételi frekvencia, vagy a buffer méret), a fordító nem minden esetben tudja

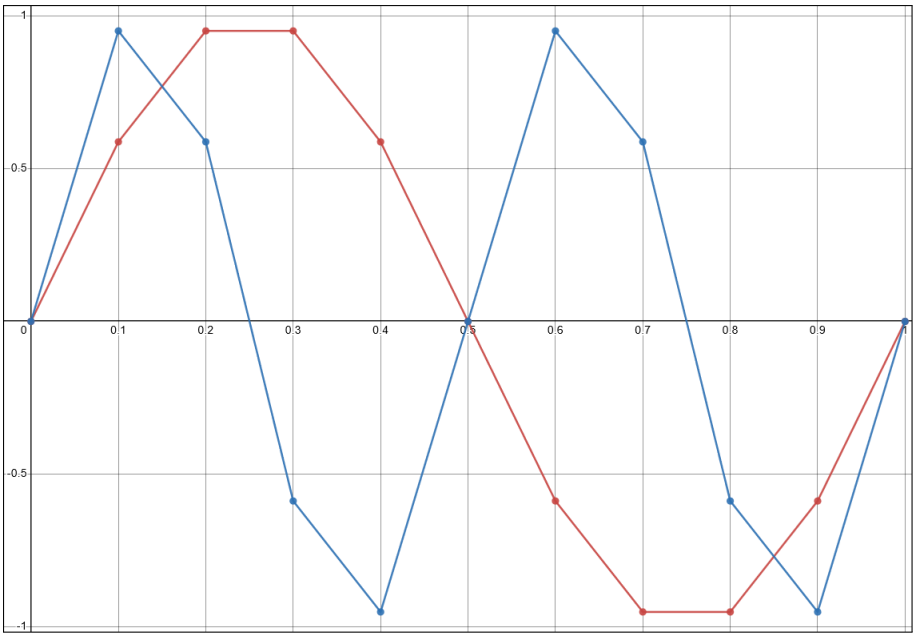
kihasználni a modern processzorokban található SIMD architektúrát sem. Emiatt az audio applikációk gyorsasága a hardver egyszálas hatékonyságán múlik.[4]

Mivel az ismételt `sin` hívás lassú lenne a szintetizátor valós idejű alkalmazásához, egy keresőtáblás megoldást alkalmaztam. Az `OscillatorParameters` osztály tartalmaz egy `juce::LookupTableTransform` tömböt, amiben az oszcillátor paraméterei által leírt hullám van tárolva `LOOKUP_POINTS` pontossággal. Ez a keresőtábla paramétermódosítások esetén frissül, hogy valós időben hallhatók legyenek a felhasználó módosításai. Ezt egy dolgozó szál végzi, ami időzítőről indul. Szálbiztonság érdekében a keresőtáblán tripla bufferelést végeztem.[3]

A keresőtáblában meghatározható, hogy hány pont reprezentálja a tárolt függvényt. Két pont közötti kérések esetén lineáris interpolációt végez a környező pontokkal. Harmonikus összetétel esetén minél nagyobb egy harmonikus frekvenciája, annál kevesebb pont reprezentálja egy teljes ciklusát.

Például: ha $\frac{n*2\pi}{T}$ frekvencia mellett, $10*n$(+1, ha figyelembe vesszük a következő ciklus 0 pontját) pont esetén egy teljes ciklust 10 minta reprezentál. Ha felveszünk egy $\frac{2n*2\pi}{T}$ frekvenciájú harmonikust, annak egy teljes ciklusára már csak 5 (+1) pont marad. Minél kevesebb pont áll rendelkezésünkre, annál nagyobb az interpolációs zaj, emiatt a legnagyobb reprezentált harmonikus függvényében kell megadni a pontok számát. Így megadható egy felső zajhatár. Ha az utolsó harmonikus egy ciklusát 64 ponttal reprezentáljuk, az interpolációs zaj kb. 50 dB-el kevesebb az adott harmonikus jelénél. Figyelembe véve, hogy a legtöbb felhasználási esetben a harmonikusok jelszintje fordítottan arányos a sorszámukkal, 64*`HARMONIC_N`+1 pont elég a keresőtáblába.

[4] A Google Benchmark tesztelőkörnyezet szerint az egyszálas tesztekben a processzorom átlagosan 30 ciklusig dolgozott egy `std::sin` híváson



3.8. ábra. Interpoláció n=1 esetén

Az általam használt `juce::LookupTableTransform` tömb egy mip-map. A tömbben minden keresőtábla fele annyi harmonikust tartalmaz, mint az előző, fele akkora pontossággal. Egy szólam az általa generált hullám frekvenciájának meghatározása után a megkeresi, hogy melyik a legnagyobb harmonikus, amit biztonságban generálhat, anélkül, hogy aliasing zaj keletkezzen a jelben, majd kiválasztja azt a keresőtáblát, aminek legnagyobb harmonikusa ezzel a frekvenciával kisebb vagy egyenlő. Aliasing zaj akkor keletkezik egy rendszerben, amikor egy olyan frekvenciájú hullámból veszünk mintákat, amit a rendszer nem tud reprezentálni. Egy rendszer által feltüntethető frekvencia-tartománynak felső határa a mintavételi frekvencia fele.[4]

A keresőtáblák legnagyobb keresőtábla szerint csúcsnormalizáltak. Így a harmonikusok amplitúdó paraméteri relatív módon működnek.

`AdditiveSynthParameters:` Ez az osztály írja le a szintetizátor paramétereit. Mivel sok elkülönülő paraméter változásairól kap visszahívásokat ez az osztály, a paraméterek egy `paramID:String value:int` párokból álló mapben vannak eltárolva. Egy paraméterváltozás esetén a bejövő `paramID`-vel keresett rekord értéke változik meg. Létezik továbbá egy pointer minden paraméter értékére is, hogy az keresés nélkül elérhető legyen a jelfeldolgozási kódban.

`AdditiveVoice:` Ez az osztály írja le a szintetizátor egy szólamajának működését. A szülő `juce::Synthesiser` a bejövő MIDI hangjegyekre indít el egy-egy ilyen

szólamot, amíg el nem éri a 32-t (`SYNTH_MAX_VOICES` paraméterrel módosítható) az egy időben aktív szólamok száma. A 33. szólam indításakor ún. note-stealing történik, amikor a szintetizátor kivágja a legrégebbi szólamot, hogy újat indíthasson.

Minden szólam a befövő MIDI hangjegyhez tartozó frekvencián kéri le a mintákat a keresőtáblából. Ezt a frekvenciát modulálják a hangoló paraméterek, valamint a MIDI pitch wheel. Az uniszónó pároknak is ez a kiindulási frekvenciája, habár ezek az uniszónó paramétereknek megfelelően elhangolható. Ezeket a funkciókat az `updateFrequencies` és az `updateAngles` függvények végzik. Frekvenciák számolása után `findMipMapToUse` keresi meg a helyes keresőtáblát a további generáláshoz.

Új MIDI hangjegy esetén a fázisok az `updatePhases`, a burkológörbe paraméterei az `updateADSRParams` függvényekkel frissülnek.

A szólam audio kimenetének kiszámolása a `renderNextBlock` függvényben történik. A szólam itt kéri le a keresőtáblából az alaphang és az uniszónók mintáit, keveri őket össze, majd alkalmazza a MIDI hangjegy hangerejét és a burkológörbét a bufferre.

`AdditiveSound:` A `juce::Synthesiser` osztály képes több hangot számon tartani. A program esetében ez egy általános hangot reprezentál, amit bármelyik MIDI csatornán és hangjegyen le lehet játszani.

`Processor::Effects`

A `Processor::Effects` névtér tartalmazza a szintetizátor második fő blokkjának osztályait:

- `EffectProcessorChain`
- `EffectProcessor`:
  - `ChorusProcessor`
  - `CompressorProcessor`
  - `DelayProcessor`
  - `EqualizerProcessor`
  - `FilterProcessor`
  - `PhaserProcessor`
  - `ReverbProcessor`
  - `TremoloProcessor`

`EffectProcessorChain:` Ez az osztály tartalmazza az effekt egységek láncát. A lánc tulajdonképpen egy tömb, amiben az osztály az egységeket sorban dolgozza fel az audio hívás során. A lánc maga is a `juce::AudioProcessor` osztályból származik, így a szintetizátorhoz hasonlóan, könnyen adaptálható.

Az effekt egységek és a bypass paraméterek atomikusan elérhetők, így cserélésük folyamán egyik elérő szál sem fog blokkolni.

Ebben az osztályban az effekt egység választó paraméterek nem automatizálhatók, mivel ezek a paraméterek megváltozása memóriafoglalással jár.

Egy időben a lánc minden típusból csak egyet tartalmazhat.

`EffectProcessor:` Az effekt egységek ősosztálya (szintén a `juce::AudioProcessor`-ból származik)

`ChorusProcessor:` A kórus effekt a `juce::Chorus` implementációt használja.

`CompressorProcessor:` A kompresszor effekt a `juce::Compressor` implementációt használja. A `juce::DryWetMixer` tag a bejövő jel és a kompresszor kimenetének keveréséért felelős.

`DelayProcessor:` A késleltetés effekt egy `juce::DelayLine` objektumot használ, mely megkapja az effekt bemenetét, és késleltetve visszaadja. Az így keletkezett kimenet szűrés után visszacsatolásra kerül ugyanabba az objektumba a feedback paraméter szerint. A `juce::DryWetMixert` tag a bejövő jel és a késleltetés kimenetének keveréséért felelős.

`EqualizerProcessor:` Az ekvalizer effekt egy 10 sávos szűrő tömb. Minden sáv egy csúcsszűrő fix közép-frekvenciával, aminek Q paramétere függ az erősítés mértékétől. Minél közelebb van az erősítés mértéke a 0 dB-hez, annál szélesebb sávra hat a szűrő.

`FilterProcessor:` A szűrő effekt első- és másodrendű Butterworth szűrők sorozata. A szűrők lehetnek aluláteresző vagy felüláteresztő szűrők. A meredekség paraméter határozza meg a szűrők rendjét:

- 6 dB/Oktáv: 1 db elsőrendű és 1 db kikapcsolt szűrő
- 12 dB/Oktáv: 1 db másodrendű és 1 db kikapcsolt szűrő
- 18 dB/Oktáv: 1 db másodrendű és 1 db elsőrendű szűrő

- 24 dB/Oktáv: 2 db másodrendű szűrő

A Butterworth szűrők előnye, hogy az átengedési sávban egyenes a frekvenciaválaszuk.[5]

`PhaserProcessor:` A fázis-modulátor effekt a `juce::Phaser` implementációt használja.

`TremoloProcessor:` A tremoló effekt egy alacsony frekvenciájú szinusz hullám mentén modulálja a bejövő jel amplitúdóját. Ha az isAutoPan változó igaz, a második csatornában ezt a szinusz hullámot eltolja fél fázissal, így a kettő mindig ellentétesen modulálja a csatornája jelét.

3.4.2. Editor

Ez a névtér tartalmazza a plugin megjelenítési kódját és annak struktúráját.

Ennek gyökere a `PluginEditor.h` és `PluginEditor.cpp` fájlokban meghatározott `VST_SynthAudioProcessorEditor` osztály. Amikor a felhasználó megnyitja a plugin felületét, a hoszt a plugin fő processzor osztályának `createEditor` függvényével hozza létre a megjeleníteni való nézetet. Mivel a nézet élettartama teljesen tetszőleges a processzor élettartamához képest, a kettő állapotát a nézet élettartamának figyelembevételével kell szinkronizálni. Erre a célra a keretrendszer paraméter-csatoló objektumokat ad: Ezek az objektumok egy felületi elemet kötnek egy paraméterhez.

A szintetizátor összes komponensének szüksége van egy referenciára, vagy magára a `VST_SynthPluginProcessor` osztályra, vagy az `AudioProcessorValueTreeState`-re, ami a paramétereit tárolja.

Új komponensek létrehozásához a `repaint` és `resized` függvények implementálása szükséges.

A szintetizátor nézete átméretezhető. Ez javarészt `juce::Grid` tárolók segítségével történik.

A `VST_SynthAudioProcessorEditor` osztálynak két gyerek-komponense van:

- A `VST_SynthTabbedComponent` komponens a szintetizátor 3 füléért felelős
- A `juce::MIDIKeyboard` komponens a felületen megjelenő billentyűzet

Tartalmaz a névtér továbbá egy megjelenésleíró osztályt és a nézet paraméterlistáját:

- `VST_SynthLookAndFeel`: A felület színösszeállítását tartalmazza

- `EditorParameters.h`: A felület globális paramétereit tartalmazza

`VST_SynthTabbedComponent`

A `VST_SynthTabbedComponent` 3 fület tartalmaz:

- `OscillatorTab`
- `SynthesizerTab`
- `EffectsTab`

`juce::MIDIKeyboard`

A `MIDIKeyboard` egy billentyűzetet jelenít meg a felületen. A billentyűzet használatához egy `juce::MidiKeyboardState` szükséges. Ez a `MidiKeyboardState` a processzorhoz tartozik. A `processBlock` függvényben a `keyboardState` MIDI üzenetei szinkronizálásra kerülnek a függvény paraméterében bejövő MIDI buffer üzeneteivel. Így a billentyűzet komponensen lehet játszani, továbbá, ha külső MIDI üzeneteket fogad a plugin, azok is meg lesznek jelenítve a billentyűzeten.

`Editor::Oscillator`, `OscillatorTab`

Az `Editor::Oscillator` névtér tartalmazza az oszcillátorhoz tartozó nézet komponenseket:

- `OscillatorTab`:
  - `WaveformViewer`
  - `WaveformEditor`:
    - `PartialSlider`

`WaveformViewer:` Ez a komponens egy oszcilloszkóp. Kirajzolja az oszcillátor által generált hullám egy teljes fázisát.

`WaveformEditor:` Ez a komponens harmonikusonként egy-egy `PartialSlider`-t tartalmaz. Ezekkel lehet a harmonikusok amplitúdójait és fázisait állítani.

`Editor::Synthesizer`, `SynthesizerTab`

Az `Editor::Synthesizer` névtér tartalmazza a szintetizátorhoz tartozó nézet komponenseket:

- `SynthesizerTab`:
  - `WaveformSelector`
  - `PhaseComponent`
  - `UnisonComponent`
  - `TuningComponent`
  - `ADSRComponent`
  - `SynthGainComponent`:
    - `GainKnob`
    - `LevelMeter`

`WaveformSelector:` Ez a komponens egy gombot tartalmaz, amit megnyomva megjelenik egy lenyíló menü, ami `WaveformMenuItem` elemeket tartalmaz. A gomb megjelenése egy `WaveformViewer`.

A `WaveformMenuItem`-ből származó alosztályoknak implementálni kell a `prepareWaveform` és `drawWaveform` függvényeket. A `prepareWaveform` függvénnyel elő lehet készíteni egy oszcillátor állapotot, ami az adott menü elem kiválasztásakor betöltésre kerül. A `drawWaveform` függvény a menü elem megjelenéséért felelős. Lehet szöveg, grafikus elem, stb.
Alapvetően a következő opciók vannak implementálva:

- Szinuszoid
- Háromszög
- Négyszög
- Fűrész
- Négyszög-Fűrész

Betöltés után az Oscillator fülön ezek a konfigurációk tovább módosíthatók.

`PhaseComponent:` A szintetizátor fázis paramétereit kezelő komponens.

`UnisonComponent:` A szintetizátor uniszónó paramétereit kezelő komponens.

`TuningComponent:` A szintetizátor hangoló paramétereit kezelő komponens.

`ADSRComponent:` A szintetizátor burkológörbéjének paramétereit kezelő komponens.

`SynthGainComponent:` Ez a komponens egy fő hangerőszabályzót (`GainKnob`) tartalmaz a szintetizátor kimenetére. Továbbá a szabályzó mellett található két `LevelMeter` komponens is, amik a szintetizátor sztereó kimeneti RMS szintjét jelzik.

`Editor::Effects, EffectsTab`

Az `Editor::Effects` névtér tartalmazza az effekt lánc és effekt egységek komponenseit

- `EffectSelectors`:
  - `EffectSelector`
- `EffectEditors`:
  - `EffectEditor`:
    - `ChorusEditor`
    - `CompressorEditor`
    - `DelayEditor`
    - `EqualizerEditor`
    - `FilterEditor`
    - `PhaserEditor`
    - `ReverbEditor`
    - `TremoloEditor`

`EffectSelectors:` Ez a komponens `FX_MAX_SLOTS` db `EffectSelector`-t tartalmaz. Az `EffectSelector` komponensekben lehet kiválasztani, hogy az adott helyen melyik effekt egység legyen betöltve, valamint, hogy az adott helyen ki legyen-e kapcsolva a feldolgozás. Amikor egy helyen valamelyik effekt egység ki van választva, a másik helyeken az az egység nem elérhető.

`EffectEditors:` Ez a komponens tartalmazza a jelenleg betöltött effekt egységek (`EffectEditor`-ból származó) komponenseit.

3.4.3. Utils

Az `Utils` névtér pár segítő struktúrát tartalmaz:

- `WorkerThread`: Egy `juce::Thread`, ami függvényt kap paraméterül és futtat
- `TripleBuffer`: A keresőtáblák szálbiztonságát biztosítja[3]

3.5. Eszközök

A tervezéshez, fejlesztéshez, teszteléshez és vizualizáláshoz a következő eszközöket használtam:

- Desmos[6]: A jelfeldolgozás optimalizálására és a matematika ellenőrzésére
- Lucidchart[7]: A diagrammok elkészítésére
- Projucer[8]: A projekt kezelésére
- Visual Studio Code[9]: Kód írására és formázására
- Visual Studio 2022[10] : A projekt építésére és a debugger futtatására
- Google Benchmark[11]: Apróbb tesztelésekre
- VST3 SDK[12]: VST validatorral való tesztelésre
- pluginval[13]: Tesztelésre
- Audacity[14]: Kimenet vizualizálására
- FL Studio[15]: Tesztelésre
- Ableton Live[16]: Tesztelésre
- Bitwig Studio[17]: Tesztelésre

4. fejezet

Tesztelés

A plugin tesztelését több lépcsőben végeztem:

1. Jelfeldolgozás
2. Plugin validáció
3. Hoszt kompatibilitás
4. Hatékonyság

4.1. Jelfeldolgozás

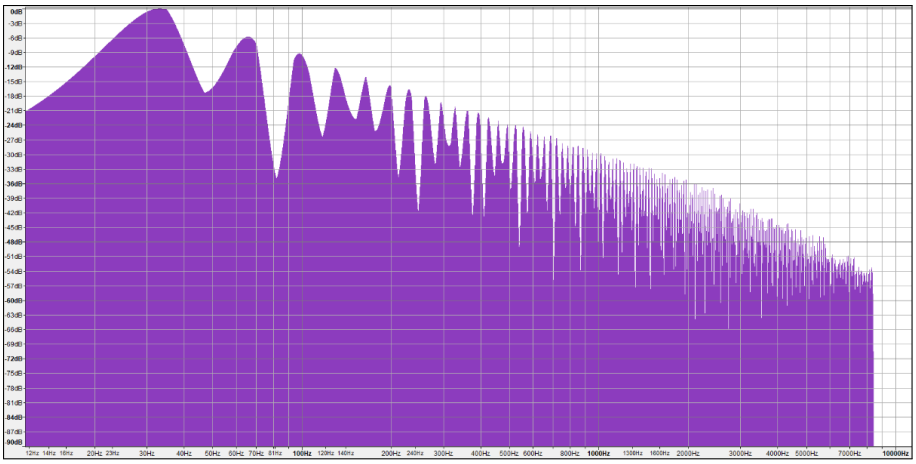
Ez a teszt a plugin paraméter és midi bemeneteire elvárt audio kimenet helyességét ellenőrzi. A tesztelést manuálisan végeztem: FL Studioban előkészítettem a tesztelési bemenetet, majd renderelés után Audacityben ellenőriztem a generált kimenetet.

4.1.1. Oszcillátor működése

Fűrész hullám

- Az oszcillátor fűrész hullámot generál
- A szintetizátor `Sustain` és `Gain` paraméterei 100%, `Attack` és `Decay` 5 ms, minden más paraméter 0
- Az effekt lánc üres

A fűrész hullám k-adik harmonikusa $\frac{1}{k}$ amplitúdóval rendelkezik.[1] Ez a kimenet spektrum-vizsgálatán is látható:



4.1. ábra. Fűrész hullám frekvencia spektruma

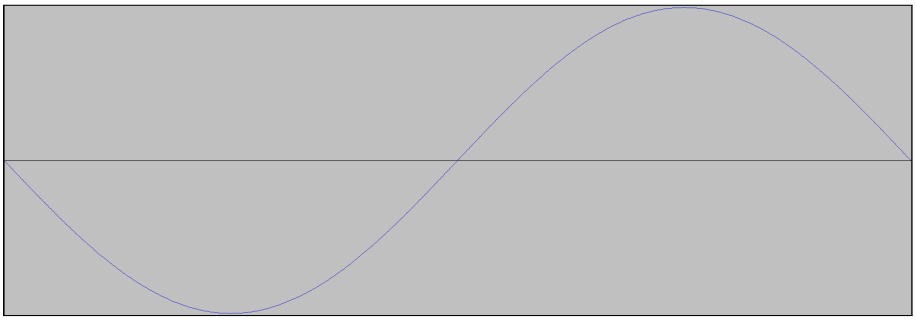
Ha a feltűntetett decibel értékeket átváltjuk, a harmonikusok amplitúdója megegyezik az oszcillátor paramétereivel.

4.1.2. Szintetizátor működése

Fázis

- Az oszcillátor egy szinuszoid hullámot generál
- A szintetizátor `Sustain` és `Gain` paraméterei 100%, a `Phase` paraméter 50%, minden más paraméter 0
- Az effekt lánc üres

A hullám fél fázissal eltolódott.



4.2. ábra. A fázis-eltolt szinuszoid hullám képe

[1] https://en.wikipedia.org/wiki/Sawtooth_wave

Uniszónó

- Az oszcillátor egy szinuszoid hullámot generál
- A szintetizátor paraméterei: `Sustain`: 100% és `Gain`: 10%, `Unison Pair Count`: 5, `Unison Detune`: 100 cent, `Unison Gain`: 100%, minden más paraméter 0
- Az effekt lánc üres

Az alaphang és az 5 uniszónó pár jelen van a kimenetben és logaritmikusan oszlanak el a frekvenciatartományban.[2]

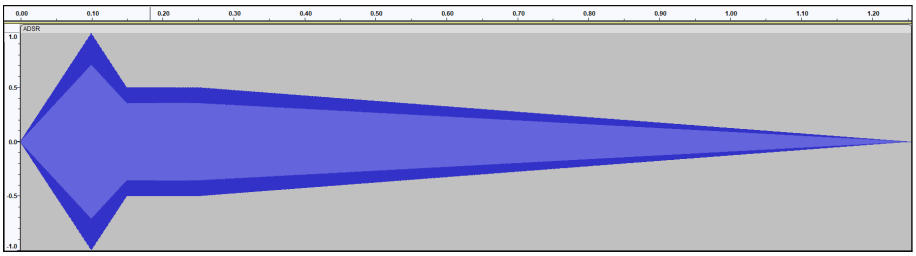
4.3. ábra. Az uniszónó spektruma (ráközelítve)

Burkológörbe

- Az oszcillátor egy szinuszoid hullámot generál
- A szintetizátor paraméterei: `Attack`: 100 ms, `Decay`: 50 ms, `Sustain`: 50%, `Release`: 2000 ms és `Gain`: 100%, minden más paraméter 0
- Az effekt lánc üres

Látható, hogy az első 100 ms alatt növekszik, majd 50 ms-ig csökken az amplitúdó, míg el nem éri az 50%-ot. A MIDI hangjegy vége után 1000 ms alatt az amplitúdó visszaesik 0-ra.

[2]Ebben a példában az Audacity spektrum analízise nem volt elég pontos, hogy látható legyen az 0.2-0.2 félhang eltérés, emiatt FabFilter Pro-Q-t használtam

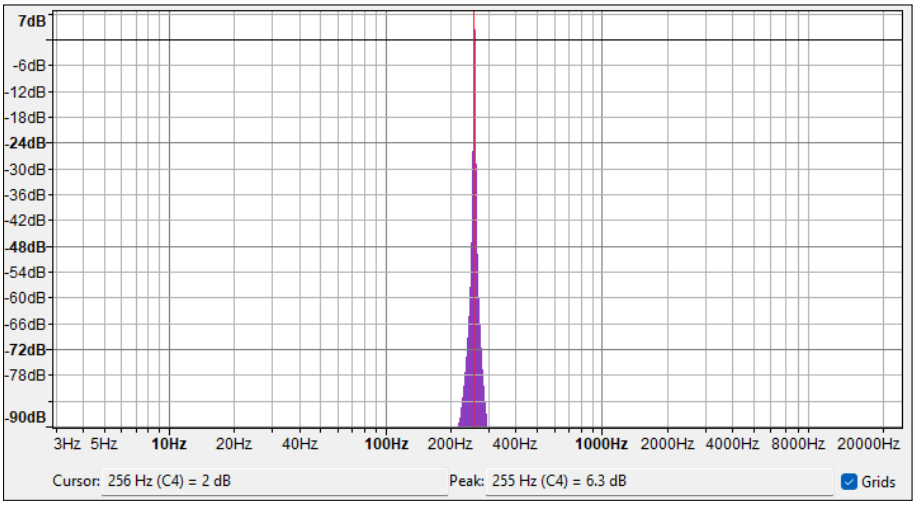


4.4. ábra. A burkológörbe kimenete

Hangolás

- Az oszcillátor egy szinuszoid hullámot generál
- A szintetizátor paraméterei: `Sustain`, `Gain`: 100%. A hangolás: -1 oktáv, +3 félhang és -46 cent. Minden más paraméter 0
- Az effekt lánc üres

A4 MIDI hangjegy esetén a jel frekvenciája 440 Hz. A paraméterek alapján a kimenet várt frekvenciája 254.8 Hz.



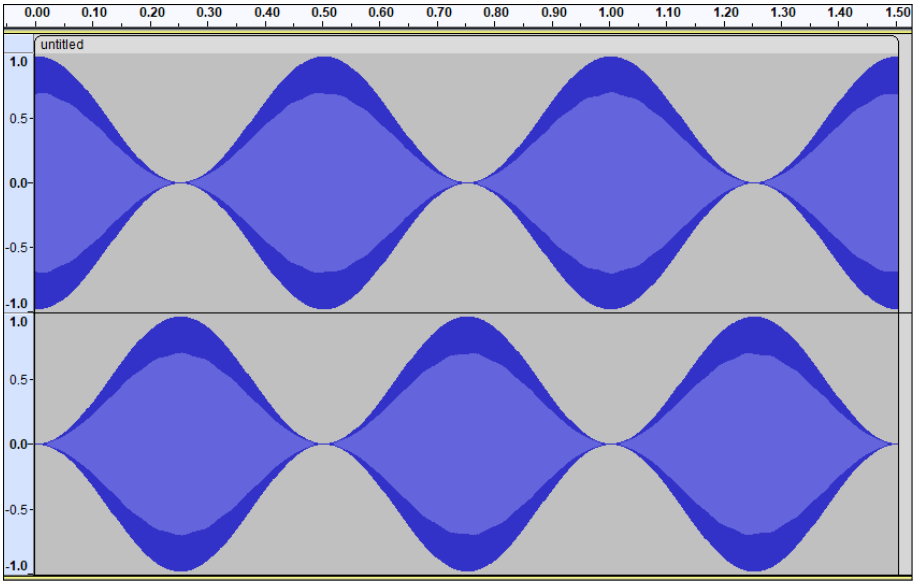
4.5. ábra. A hangolás kimenete

4.1.3. Effektek működése

Tremolo

- Az oszcillátor egy szinuszoid hullámot generál
- A szintetizátor paraméterei: `Sustain`, `Gain`: 100%, minden más paraméter 0
- Az effekt láncban egy Tremolo egység van használatban.
- Tremolo paraméterei: `Depth`: 100%, `Rate`: 2 Hz, `Auto-Pan`: Be

A Tremolo elvárt működése, hogy a két sztereó csatorna tartalmának erősítését egy 2 Hz frekvenciájú szinuszoid hullámmal modulálja. A két csatorna modulációja ellentétes egymással.



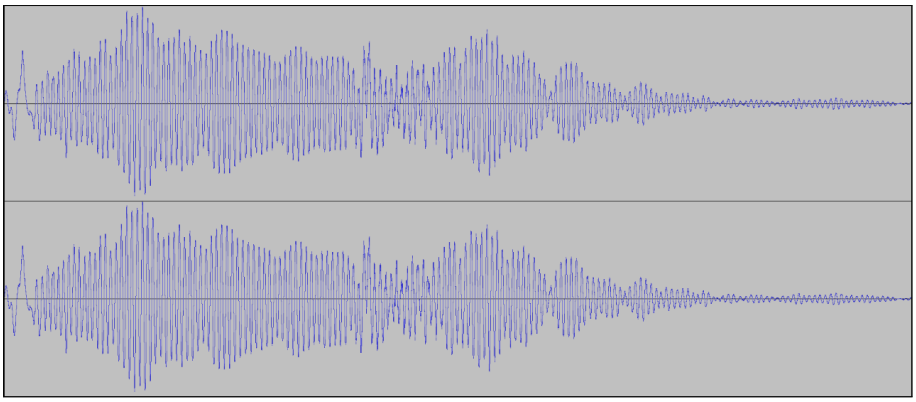
4.6. ábra. Tremolo kimenete

Effektek sorrendje

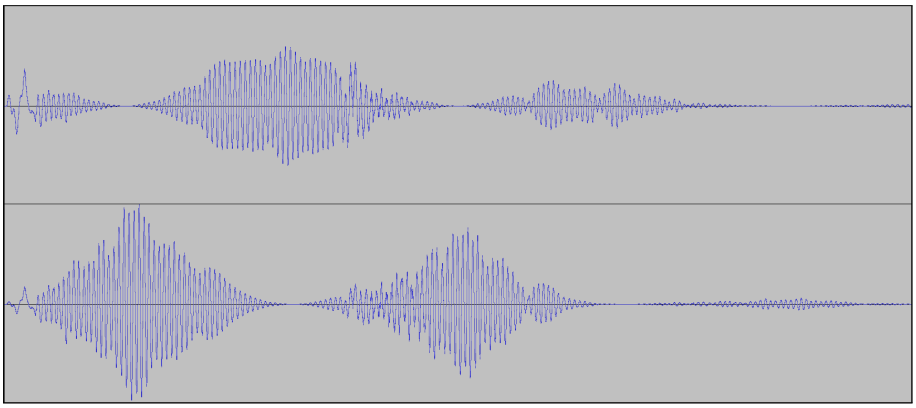
Ez a teszt az effektek lehetséges sorrendjei közötti különbséget demonstrálja

- Az oszcillátor egy szinuszoid hullámot generál
- A szintetizátor paraméterei: `Sustain`, `Gain`: 100%, minden más paraméter 0
- Az effekt láncban egy Tremolo és egy Reverb egység van használatban. A két eset között a kettő egymáshoz viszonyított sorrendje változik.
- Tremolo paraméterei: `Depth`: 100%, `Rate`: 2 Hz, `Auto-Pan`: Be
- Reverb paraméterei: `Wet`: 100%, minden más paraméter 0

A Reverb egység mono hangot generál ebben a konfigurációban, míg az Tremolo sztereó modulációt végez. Ha a Reverb effektet dolgozzuk fel előbb, akkor a várt kimenetben eltér a két csatorna tartalma, a fordított esetben nem.



4.7. ábra. Tremolo -> Reverb kimenete



4.8. ábra. Reverb -> Tremolo kimenete

4.2. Plugin validáció

A plugin validációja általános, plugin formátummal kapcsolatos automatikus tesztelés.

Ezt a tesztet a pluginval[13] és a VST SDK3 validator[12] eszközökkel végeztem. A VST3 SDK validator a plugint a specifikációja szerint teszteli.
A pluginval több plugin formátumot támogat. Ezek általános funkcionalitását teszteli. pl.:

- Paraméterek szálbiztonsága
- A felület függetlensége az audio feldolgozástól
- A ki és bemenet konfigurációk
- Véletlenszerű paraméterek hatása
- Különböző mintavételi frekvencia és buffer méret konfigurációk
- Állapot mentése és visszaállítása

A szintetizátor átment az összes teszten, amit ezek az eszközök futtattak.

4.3. Hoszt kompatibilitás

Ezek a tesztelések az alapvető hoszt-plugin kommunikációt foglalják magukba. A plugin kompatibilitását a következő 3 hoszton teszteltem:

- FL Studio[15]
- Ableton Live[16]
- Bitwig Studio[17]

Teszteltem:

- A MIDI bemenet fogadását
- A paraméterek automatizációját
- Az állapot mentését és betöltését
- A felület működőképességét

Egyik hosztban sem tapasztaltam hibás vagy másképp nem kívánatos működést.

4.4. Hatékonyság

A hatékonysági tesztet egy rövid stressz-tesztnek szántam. Megnéztem, hogy a plugin összes funkciójának használata közben a bufferelési idő mekkora részét tölti a processzorom (a lehető legrosszabb körülmények között) a buffer kiszámításával.

Ez nem egy pontos teszt, de egy jelzést adott arról, hogy mennyire használható a program egy valódi rendszeren.

A teszt körülményei:

- Mind a 32 szólam egyszerre kell szóljon
- Az uniszónó minden paramétere használatban kell legyen
- A fázis randomizálva van
- Minden effekt egység be van töltve nem 0 paraméterekkel
- A processzor alacsony órajelen fut (aktív állapotban 800 MHz-1600 MHz)

A teszthez FL Studio Plugin performance monitor eszközét használtam. Az eszköz időnként mintát vesz a jelenleg feldolgozás alatt lévő függvényhívásokból az aktív pluginek között. Hosszú távon így kiszámítható egy plugin átlagos munkaideje. Három darab egy perces teszt alatt a fenti tesztkörülmények mellett a plugin nagyjából a rendelkezésére álló idő 1/3-át használta fel.

Ez az eredmény az alsó határértéke a plugin hatékonyságának egy messze nem ideális rendszeren. Egy általános felhasználási esetben nem fog minden szólam egyszerre szólni és nem lesz minden effekt egység egyszerre használatban.

5. fejezet

Továbbfejlesztési lehetőségek

A kreatív felhasználást célzó, jelfeldolgozáson alapuló alkalmazások lehetséges bővítésének határa a csillagos ég.

Tulajdonképpen bármilyen fejlesztést el lehet végezni, amíg a program nem veszít a valós-idejű tulajdonságából. A jelenlegi digitális audio plugin piac viszonylag telített, mivel a jól megalapozott jelfeldolgozási módszerek alkalmazásai visszanyúlnak az analóg zenei jelfeldolgozás korszakába. Úgy gondolom, hogy a jelenlegi fellendülés a gépi tanulási területeken új lendületet adhat az iránynak, de ez csak egy kitekintés az audio alkalmazások szélesebb területe felé.

Az én esetemben tervezés és fejlesztés közben felmerült több potenciális funkció, ami idő hiányában vagy egyéb okokból nem került a végső alkalmazásba:

- Jelenleg beállított hullámforma mentése és betöltése fájlból: Ezt két új `WaveformSelectorItem` implementálásával lehetne megoldani
- Audio fájlon elvégzett Fourier analízissel kinyert harmonikusok: Szintén `WaveformSelectorItem`, ami egy audio fájlt nyit meg és dolgoz fel. Ezzel a fájl tartalmát lehet periodikusan szintetizálni
- Szűrő a szólamokon: Számításilag viszonylag drága, de egy szólamonkénti szűrőre lehet a szólamban burkológörbét csatolni, vagy a szűrési frekvenciát a leütött billentyűvel modulálni.[1]
- Többféle választható oszcillátor beépítése

[1] Minimoog D szintetizátor CV/Gate modulált szűrőihez hasonlóan

- Egy második oszcillátor beépítése saját szólamokkal (és szűrőkkel a szólamokon)
- Zajgenerátorok
- Kétdimenziós keresőtáblák használata: Így egy teljes hullámtáblás szintetizátort kapunk, ami több beállított hullámot is tud egyszerre tárolni, és interpolációval átmenetet képezni közöttük.
- Több effekt egység lehetősége: wave shaper, konvolváló
- Több szűrő opció: shelving, sáváteresztő, sávzáró szűrő
- Beépített modulációs mátrix: modulációs forrás és modulációs célpont párokat tartalmaz. Egy forrás lehet egy alacsony frekvenciájú oszcillátor (LFO), burkológörbe, MIDI CC vagy egy másik paraméter, a célpont pedig egy paraméter (ami lehetne a moduláló forrás paramétere is pl.: az LFO frekvenciája)
- A `juce::LookupTableTransform` osztály lineáris interpolációja helyett egy pontosabb interpoláció használata.
- A `juce::LookupTableTransform` használata helyett az additív szintézist meg lehet közelíteni trigonometriai iterációs feladatként is.[18]

6. fejezet

Összegzés

Már az egyetemre jelentkezéskor tudtam, hogy valamilyen audio applikációt szeretnék fejleszteni szakdolgozatomnak. Saját tapasztalataimmal indultam neki ennek a feladatnak. Feltettem magamnak a kérdést: „Mit várnék egy VST-től, ha használni is akarnám egy zeneszerkesztési munkafolyamatban?". Erre a kérdésre alakult ki bennem először, hogy pontosan mit is szeretnék elérni.

A program tervezése, a keretrendszer és környezet megismerése és a hibák javítása közben sokat tanultam az információs rendszerek működéséről, a jelfeldolgozás jelentőségéről, valamint ennek a feladattípusnak a nehézségeiről. Úgy gondolom, hogy sokat fejlődtem az utóbbi hónapokban, de ez csak a felszínt súrolja ebben a témában.

Ezúton szeretnék köszönetet mondani konzulensemnek, Dr. Biró Csabának, aki hozzáértésével és türelmével segített engem tájékozódni ebben a folyamatban. Továbbá köszönöm a családomnak és barátaimnak, hogy támogattak tanulmányaim során.

7. fejezet

Mellékletek

- Témabejelentő

  Habzda_Bálint_VP8EM3_Témabejelentő.pdf

- Eredetiség Nyilatkozat

  Habzda_Bálint_VP8EM3_Eredetiség_Nyilatkozat.pdf

- projektet és kódot tartalmazó mappa

  VST_Synth

- A plugin osztály diagrammja

  Plugin Class Diagram.pdf

- Szekvencia diagramm a plugin betöltéséről

  sd Plugin Loading.png

Irodalomjegyzék

[1] *VST3 Dokumentáció*. https://steinbergmedia.github.io/vst3_dev_portal/pages/Technical+Documentation/API+Documentation/Index.html. Elérve: 2023-05-05.

[2] *JUCE dokumentáció a paraméter visszahívásokról*. https://docs.juce.com/master/classAudioProcessorListener.html#aa08a28d958759256aecd0f4af8a826f4. Elérve: 2023-05-05.

[3] kamedin. *Tripla bufferelés*. https://forum.juce.com/t/parameter-changes-thread-safety/50098/14. Elérve: 2023-05-04. 2022.

[4] A.V. Oppenheim és R.W. Schafer. *Digital Signal Processing*. Prentice-Hall, 1975. ISBN: 9780132146357. URL: https://books.google.hu/books?id=vfdSAAAAMAAJ.

[5] G. Ballou. *Handbook for Sound Engineers*. Taylor & Francis, 2013. ISBN: 9781136122545. URL: https://books.google.hu/books?id=S4nBNZ_EJwwC.

[6] *Desmos*. https://www.desmos.com/. Elérve: 2023-05-04.

[7] *Lucidchart*. https://www.lucidchart.com/pages/. Elérve: 2023-05-04.

[8] *Projucer*. https://juce.com/. Elérve: 2023-05-04.

[9] *Visual Studio Code*. https://code.visualstudio.com/. Elérve: 2023-05-04.

[10] *Visual Studio 2022*. https://visualstudio.microsoft.com/vs/. Elérve: 2023-05-04.

[11] *Google Benchmark*. https://github.com/google/benchmark. Elérve: 2023-05-05.

[12] *VST3 SDK*. https://github.com/steinbergmedia/vst3sdk. Elérve: 2023-05-05.

[13] *pluginval*. https://github.com/Tracktion/pluginval. Elérve: 2023-05-05.

[14] *Audacity*. https://www.audacityteam.org/. Elérve: 2023-05-05.

[15] *FL Studio*. https://www.image-line.com/. Elérve: 2023-05-05.

[16] *Ableton Live*. https://www.ableton.com/en/. Elérve: 2023-05-05.

[17] *Bitwig Studio*. https://www.bitwig.com/. Elérve: 2023-05-05.

[18] W.H. Press és S.A. Teukolsky. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Numerical Recipes: The Art of Scientific Computing. Cambridge University Press, 2007. ISBN: 9780521880688. URL: https://books.google.rs/books?id=1aA0dzK3FegC.

Ábrák jegyzéke