BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR
MÉRÉSTECHNIKA ÉS INFORMÁCIÓS RENDSZEREK TANSZÉK

Demjén Ádám

Orgonahang-szintézis megvalósítása VST környezetben

KONZULENS
dr. Sujbert László
docens

2009

Melléklet

Orgonahang-szintézis megvalósítása VST környezetben

Az utóbbi években több sikeres önálló laboratóriumi feladat, TDK dolgozat, illetve diplomaterv született melodikus hangszerek (orgona, zongora, hegedű stb.) hangjának digitális szintézise témakörben. Az elméleti eredmények valós idejű rendszeren történő implementálása azonban gyakorta váratott magára, elsősorban azért, mert nem állt rendelkezésre elégséges számítási kapacitás. A ma már hétköznapinak számító személyi számítógépek is alkalmasak azonban hangszintézis-eljárások valós idejű futtatására, még a különösen számításigényes fizikai modellek is sikerrel implementálhatók.

Orgona hangjának szintézisére született egy jól kidolgozott, az orgonahangzás valósághű szimulációját lehetővé tevő eljárás. A modell az ún. additív szintézis hagyományos megközelítéséből kiindulva, az orgonasípok hangjának periodikus jelmodell alapú szintézisét valósítja meg, kiegészítve a sípzaj, valamint a felfutási és lecsengési tranziensek modellezésével. A modell paramétereit orgonasíp-felvételek elemzésével határozták meg.

A személyi számítógépen történő implementációhoz a hozzá kapcsolódó, MIDI interfésszel ellátott billentyűzet, valamint a számítógépen futó VST környezet áll rendelkezésre.

Fentiek alapján a szakdolgozat-készítés keretében az alábbi konkrét feladatokat kell megoldani:

- Mutassa be az orgona hangjának valós idejű szintézisére alkalmas digitális modellt!
- Az irodalom alapján készítse el a VST környezetben megvalósítható, MIDI billentyűzettel, illetve MIDI fájl segítségével vezérelhető orgonahang-szintetizátor rendszertervét!
- Implementálja a modellt és működését zenei példákkal demonstrálja!

dr. Sujbert László
docens

Nyilatkozat

Alulírott, ***Demjén Ádám***, a Budapesti Műszaki és Gazdaságtudományi Egyetem hallgatója kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, és a szakdolgozatban csak a megadott forrásokat használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

.....

Demjén Ádám

Kivonat

„Az akusztikus hangszerek ideje lejárt!” – hallhattuk még 10 éve ezt az elég drasztikus és merész kijelentést, de valójában az analóg és digitális szintetizátorok előretörése már korábbi időkre vezethető vissza. A virtuális szintetizátorok megjelenése ugyan tényleg új lehetőségeket nyitott, új irányvonalak keletkeztek a zenében, napjainkra divattá vált analóg hangszerek hangjának modellezése és virtuális megvalósítása. Mára világossá vált, hogy az elektronikus zene sosem volt az akusztikus zene konkurenciája, hanem végtelenül kreatív kiegészítője és folytatása, mely új dimenziókat nyitott meg a hangszerek világában. A régi, de forradalmi Moog szintetizátortól kezdve a templom ékéül szolgáló orgonáig terjedő hangszerek egyaránt csak színesítik mind az elektronikus, mind a más zenei irányzatokat.

A Virtual Studio Technology nyílt forráskódú fejlesztői készletének köszönhetően gyorsan elterjedtté váltak az otthoni és kisebb vállalkozás szintjén egyre inkább virágkorukat élő VST szoftverek. Többek között ennek köszönhető, hogy a zeneipar amatőr és professzionális szintű rétege egyaránt könnyebben hozzájuthat olyan effektekhez, szoftver szintetizátorokhoz, amelyeknek hardveres változatai nagyon drágák, vagy egyáltalán nem is léteznek még.

Dolgozatomban az orgona szintéziséhez vizsgáltam az orgonahang periodikus jelmodelljét, amelyhez nagy segítséget nyújtottak korábbi hangszintézis témát tárgyaló diplomatervek, TDK dolgozatok, valamint egy olyan speciálisan orgonával foglalkozó diplomaterv, amely részletességgel végez analízist különböző orgonamodelleken, és eltárolja az orgonahang szintézishez szükséges sztochasztikus, tranziens és más paramétereit.

A dolgozat első része áttekinti összefoglaló jellegűen a zenei hangok pszichoakusztikai jellemzőit, az orgona rövid ismertetését, az orgonahang analízisból eredő zenei hang szintézisének lehetséges fajtáit.

A második fejezetben MATLAB segítségével egy orgonahang digitális jelmodell alapú szintézise követhető nyomon, amellyel könnyen szemléltethetőek a hangszer hangjának jellegzetes tulajdonságai, matematikai leírása, majd ennek megvalósítása. Ez a szakasz az egyes lépések vizuális nyomonkövethetőségére is szolgál.

A harmadik főbb fejezetben egy viszonylag új, de annál elterjedtebb – valós idejű szintézishez kiválóan alkalmazható – fejlesztői környezetet, a VST szabványt, valamint ezzel kapcsolatosan a MIDI adatkezelést és feldolgozást vezettem be. A környezet rövid bemutatását követően a témakör a VST polifonikus orgonahang szintetizátor dokumentációjával folytatódik, bemutatva a szintézishez felhasznált digitális modellt, a fejlesztés lépéseinek részletes kifejtésével.

A tesztelés és összefoglalás fázis összegzi az elért eredményeket, értékeli a digitális jelmodellt, rávilágít a javítási felületekre és a továbbfejlesztési lehetőségekre.

Abstract

”The time of the acoustic intsruments has ended” – a quite drastic and bold statement we heard 10 years ago. Actually, the breakthrough of the analog and digital synthesizers had started even earlier. The release of the virtual synthesizers brought new possibilities and new directions in music, the modeling and virtual realisations of analog instruments came in fashion. To this day it has become clear that electronic music has never been a concurrency of acoustic music, but an excessively creative complement and continuation, which opened new dimensions in the world of instruments. From the revolutionary Moog synthesizer to the church organ, all instruments equally contribute to the diversity of music.

Thanks to the open source developing environment of Virtual Studio Technology, the VST softwares quickly became popular, they flourish on the level of small companies. This technology allows professional and amateur musicians to obtain audio effects, software synthesizers more easily. Hardware versions of such effects or virtual instruments are very expensive or don’t even exist.

In my thesis I examined the periodic signal model of the organ sound to synthesize the instrument. Previous theses and TDK papers dealing with sound synthesis offered me great help in this proccess, especially one particular thesis, that is about detailed tests of different organ models and recordings of their variate, transient and other parameters necessary to the synthesis.

The first part of the thesis reviews the psychoacoustic characteristics of sound, contains a short exposition of the organ, and reviews the possible ways of sound synthesizing derived from the analysis of the sound of the organ.

The second chapter features a simple synthesis performed with Matlab based on a digital signal model of an organ sound. This demonstrates the typical characteristics of the sound of the instrument, its mathematical representation, then its realisation. This section also features the visual representation of each step.

In the third main chapter I introduce a relatively new, but very popular developing environment, the VST standard, which is very suitable for real-time synthesis. In connection with that I also introduce MIDI data-handling and data-processing. After a short presentation of the environment, the topic continues with the documentation of the VST poliphonic organ synthesizer, bringing forward the digital model used for the synthesis, with detailed exposition of the steps of development.

The testing and conclusion phase summarizes the accomplished results, evaluates the digital signal model and reveals the possibilities of improvement and correction.

Tartalomjegyzék

Előszó

Az ember állandó egyszerűsítésre, praktikusságra, új ismeretek keresésére való törekvése az élet minden területén megmutatkozik. Ez a törekvés a forrása minden újításnak, találmánynak. Nincs ez másképpen a zene, valamint a hangszerek világában sem. A meglévő eszközeink (beleértve a hangszereinket is) mindennapos, korlátok nélküli használatának lehetővé válása a célunk. Ezeket a korlátokat számoljuk fel jelen korunkban is évről évre, mikor például a vezetékes telefont felváltottuk mobiltelefonokra, vagy mikor kiléptünk a vezetékes számítógépek és internet világából a hordozható pc-k, és vezeték nélküli internetkapcsolatok világába. Nem volt ez másképp a régmúltban sem. Nem tarthatott sokáig tehát annak az ötletnek a felszínre jövetele sem, hogy miként lehetne egy olyan, méreteiben és sokszínűségében verhetetlen hangszer, mint az orgona olyan hasonmását megalkotni, amely egyrészt kisebb, olcsóbb, másrészt minőségben is elfogadható, ezáltal elérhetővé válna szélesebb tömegek számára is. Az első ilyen sikeres próbálkozás a 30-as években a sokak számára is ismert Hammond–orgona volt. Ez a mechanikus, analóg áramkörökkel megvalósított hangszer az eredetivel akart versenyre kelni, ezzel akarták helyettesíteni a drága templomi orgonákat, valamint olcsó alternatívákat jelentett szalonokban, iskolákban, klubokban, stadionokban is.

Az alapkoncepció, hangszerek különböző modell alapú megvalósításának vezérelve az idők folyamán mit sem változott. Egymással versengtek a méretekben egyre kisebb, hangminőségben egyre jobb szintetizátorok is, mígnem elérhetővé vált a hordozhatóság is. Napjainkra már nagyon széles a választék mind analóg, mind digitális szintetizátorok terén is, a 90-es évek közepén napvilágot látott az első fizikai modellezésen alapuló digitális szintetizátor (a Yamaha cég és a Stanford-egyetem közös munkássága révén), és a korábban is hangszermodellezéssel foglalkozó cégek is áttértek a fizikai modell alapú szintézisre. A fizikai modellezés tehát nagy részletességű képet nyújt a hangszerben történő hang keletkezésének matematikai leírásáról. A hang keletkezésének fizikai modelljét alapul vevő, de magát a hangjelet modellező szintézis megvalósítását tűzte ki célul ez a dolgozat, ami az orgona hangszer hangjának vizsgálatával kezdődik, majd az akusztikus hangszer hangjának érzeti jellemzői alapján történő részekre bontása utáni egységek tanulmányozása után, (amelyet korábbi dolgozatok sikeresen véghezvittek) fokozatosan áttér ezen komponensek egyesítésére (tehát az analízis vizsgálat után a szintézisre), amely egy orgona VST szintetizátor megalkotásában csúcsosodik ki.

Az 1. fejezetben a dolgozat a zenei hangok, de főként az orgona hangjának pszichoakusztikai jellemzőivel foglalkozik, definiálva a pszichoakusztika fogalmát, a hangszereket több szempontból csoportosítja, kitér a tranziens folyamatokra, majd rövid betekintést nyújt az általános sztochasztikus jelenségekről, majd a hangszer hangját színesítő, külső közeg által gyakorolt hatásokról. A fejezet elején az orgona mint hangszer felépítéséről van szó, áttérve az orgona hangjának analízisére, pszichoakusztikai jellemzőire.

A 2. fejezet az orgona szintézisének modelljével foglalkozik. Először bemutatja az orgona hangjában érdekelt főbb szintézismódszereket történelmi példákon keresztül. Bemutatásra kerül az additív, szubtraktív, hullámtáblás szintézis, majd a fizikai-modell alapú szintézis. Ezekre az ismeretekre épülve felépít egy jelmodellt, amellyel az orgonahang szintézise történt az ezt lehetővé tevő MATLAB környezetben. Az orgona hangjának hangérzeti szempontból összetevőkre bontása után kapott komponensek részegységenkénti megvalósításának lépései nyomon követhetőek a fejezetben, amelyek könnyed ellenőrzését

ábrák teszik lehetővé. Az így előállt harmonikusok, tranziens burkológörbék és sztochasztikus jelenségek összegzésével egy néhány másodperces orgonahang *.wav* fájlformátumba történő generálása történt meg.

A 3. fejezet az orgonaszintetizátor kialakításáról, megvalósításáról, majd dukumentációjáról szól. A fejezet elején megismerteti az olvasóval a MIDI-protokoll kialakulásának okát, majd magát a bitfolyam alapú aszinkron, soros kommunikáció szabályait. Ezután a Steinberg nevű cég által kidolgozott Virtual Studio Technology bevezetése következik, amely néhány fontos alapdefiníció után (VST plugin, VST Instrument, VST Host, VST SDK) rátér a szintetizátor C++ fejlesztési nyelven írt részletes dokumentációjára, amelyet az orgonahang-szintetizátor rendszerterve előz meg. Szó lesz tehát arról, hogyan hozhatunk létre VST-ben harmonikusok dinamikus generálását végző oszcillátorokat, tranziens-burkológörbét, additív orgonazajt generáló szűrőket, a szűrőparaméterek billentyűnkénti fájlból történő beolvasását, és a polifónikus hangszer előállítását a MIDI-kezelést végző függvények segítségével.

A tesztelés fázisa a 4. fejezetben történik, ahol egy mérési összeállítással a tesztelés végrehajtható a létrehozott pluginon, rámutatva az elért eredményekre, a kijavított és esetlegesen továbbra is fellépő hibákra, és az egyszerűsítésekre. Az 5. fejezet áttekintő képet nyújt a félévben elvégzett munkákról és eredményekről. Kitér a megoldatlan hibákra, az esetleges megoldási és a szoftver továbbfejlesztési lehetőségeire. A dolgozat lezárását a függelék ábrái, képei, szoftver osztálydiagramja és útmutatásai végzik a 7. fejezetben, amelyet az Irodalomjegyzék előz meg (6. fejezet).

E sorokat szeretném megragadni, hogy köszönetet mondjak azoknak, akik nélkül ez a dolgozat nem jöhetett volna létre.

Elsőként köszönetet mondok konzulensemnek, dr. Sujbert Lászlónak, aki szakmai és szerkesztői tanácsokkal egyaránt fáradhatatlanul segítette a szakdolgozat létrejöttét.

Köszönet illeti dr. Bank Balázst, aki bevezetett a zenei jelfeldolgozás alapjaiba és konzultációs lehetőségekkor is rengeteg hasznos tanáccsal látott el.

Köszönetet mondok Sápi Dénesnek, testvéremnek Demjén Gergőnek, Munkácsi Editnek és Szalontai Péternek, akik hasznos tanácsokkal láttak el a dolgozat szerkezetét tekintve.

Végül a legnagyobb köszönet illeti szüleimet, akik az egyetemi éveim alatt a tőlük telhető legnagyobb mértékben támogattak

1. Orgona hangjának analízise, áttekintés

1.1 Az orgona felépítése

Az orgonát funkciói szerint többféle részegységre bonthatjuk. Az 1. ábrán látható az orgona szerkezeti felépítése. Minden hangszerben vannak vezérlést lehetővé tevő részek, és kimenetet produkáló, a hang keletkezéséért felelős részek. Az orgona hanggenerátorai a sípok. Ezek megszólaltatásáért az áramló levegő (orgonairodalomban szél), valamint a megszólaló sípok kiválasztásáért felelő szelepek. A megszólaláshoz szükséges levegőt a mai orgonákon villanymotoros fújtató állítja elő, amely az ehhez kapcsolódó levegőrendszeren (szaknyelven szélcsatornákon) keresztül jut a sípokhoz.

A sípok vezérlését a játszóasztal végzi. Az ezen elhelyezkedő billentyűzet határozza meg, hogy melyik szelep nyisson ki, ezáltal engedélyezhető és letiltható a sípok megszólaltatása. Egy játszóasztalon egytől kilencig terjedhet a billentyűzetsorok száma, de a temploni orgonák között a négysoros a legjellemzőbb. A kézzel játszható billentyűzetsorokat manuáloknak – ezek egymás fölött lépcsőzetesen helyezkednek el – a lábbal működtetett billentyűsorokat pedig pedáloknak nevezzük. Ez a lábnál elhelyezett nagyobb méretű hézagosabb rudak sorozata. Egy manuálon a billentyűk száma 56-61, a pedálon 30-32.[1]

A sípok vezérlését a traktúra továbbítja a szelepekhez. Ez a játszóasztalt és az orgonaszekrényt összekötő szerkezet, mely a két szerkezet között továbbítja az információkat. Kezdetben mechanikus traktúrákat alkalmaztak[1] (az orgonisták ma is ezeket kedvelik jobban, mert közvetlen kontaktust tesz lehetővé az orgonista, és a hangszer között), később azonban az elektromos traktúra terjedt el, amely a szelepnyitó relét kapcsolgatja a billentyűk segítségével. Az orgonaművészek szerint a billentés erősségének információját a mechanikus traktúra képes továbbítani, de ezt eddig a mérések nem bizonyították.

Az orgona hangkeltői, a sípok több szempont alapján sorolhatók csoportokba. A legkézenfekvőbb azonban a regiszterek szerinti csoportosítás. Egy billentyűzetsoron lévő billentyűzethez, vagy pedálhoz általában egy (ritkább esetben több) síp tartozik. Egy regiszter sípjainak használatához a regiszterkapcsolókkal való kiválasztásuk szükséges, tehát az egyes regisztereket választják ki a regiszterkapcsolók, így akár az összes regiszter is szólhat egy időben (de csak azok a hangok, amelyeket a billentyűvel engedélyezünk). Adott regiszteren lévő sípok hangérzeti szempontból hasonlóak, vagyis egy regiszter általában azonos hangszínű sípok palettáját tartalmazza[2]

Az orgonának kétféle sípfajtáját különböztetjük meg: A nagyobb számban előforduló ajaksípokat, valamint a kisebb számú nyelvsípokat. Az egyes regiszterek azonos billentyűhöz (hangmagassághoz) tartozó sípjai hangkarakterben és valós frekvenciában is eltérhetnek egymástól [1]. A regiszterek nevei jelentéssel rendelkeznek, mint például a hangszínre utaló jelzővel, vagy a síp anyagával, hangmagasságával, vagy fizikai megvalósításával elnevezett

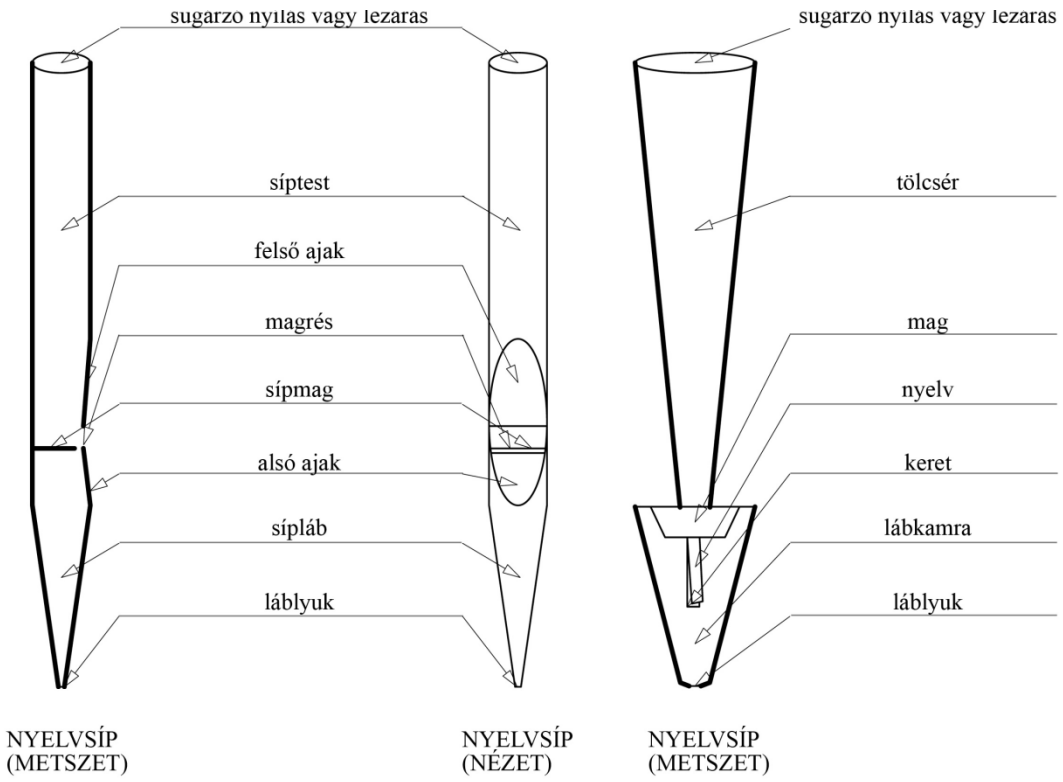
[1] Barokk orgonáknál volt jellemző, forgó áttételek, és húzólécek által teremtették meg a kapcsolatot a vezérlés,

[2]Ennek ellenére egy regiszter sípjai a hangmagasság, méret, anyag, hangszínben egyaránt eltérhetnek egymástól, ugyanis az orgonakészítők a kiegyenlített hangzásért ezeket a hangszín-változtatásokat is kieszközölik a regiszteren belül.

regisztertípusok. A nevek után lábszámmal is el szokták látni őket. A lábszám jelöli, hogy a megszólaló síp hangja milyen viszonyban van az alapfrekvenciával. Például a Princip8' síp hangmagassága az egyvonalas „A” hangnak felel meg, tehát 440 Hz-es frekvenciával szól. A 4'-as síp egy oktávval mélyebbi, a 16'-as egy oktávval magasabbi hangot jelent a fenti hangtól. Az 5 1/3' lábszámú síp a 8'-astól egy kvinttel feljebb helyezkedik el (aminek frekvenciája 660 Hz, ha ugyanazt a billentyűzetet nyomjuk le) . A 8'-asál mélyebb regiszterek jellemzően pedál által vezéreltek, az alapregiszteréknél magasabb regisztereket pedig felhangregiszteréknek szokás nevezni.

1.2. Az ajaksípok

Az ajaksípokban a fújtatóból érkező, szeleppel engedélyezett levegő a síplábon és a keskeny magrésen keresztül kiáramlik (létrejön egy levegőnyelv), és a felső ajakba ütközve, ékhangot (peremhangot) generál [2], hasonló hangkeltési jelenséggel, mint a furulyában keletkező hang. Az 1. ábrán egy ajaksíp és egy nyelvsíp részei láthatóak.



1. ábra: Ajak- és nyelvsípok részei

Az ajaksípok készülhetnek fémből vagy fából, és méretük néhány centiméterestől a 10 méteresig terjedhet. Legtöbbször hengeres formájúak, de vannak ún. kónikus, felfelé szűkülő sípok is. A fasípok lapokból vannak összeállítva, négyzetes hasáb formájúak. Az orgonák készítésénél használnak nyitott, valamint fedett ajaksípokat is, az utóbbiak egy oktávval mélyebb hangot adnak ki. A fedett fém ajaksípok kupakja lehet zárt, vagy nyitott csőtoldalékkal ellátott, hangszínt módosító hengeres (cilindrikus) rátét, vagy felfelé csúcsosodó alakú, felül lyukas sapka. Ez utóbbiak a félfedett sípok.

Az ajaknyílás mérete tapasztalati úton alakult ki az évszázadok alatt, melynek paraméterei csak szűk tartományban mozoghatnak. A hangszín változtatását tehát a síp (rezonátor) típusa (cilindrikus, kónikus, nyitott, zárt), és a síp méretei határozzák meg. Egy nyitott síp akusztikai modellje a mindkét végén sugárzó (nyitott) egydimenziós akusztikai tápvonal, a fedett síp pedig egyik végén zárt egydimenziós csőként modellezhető. A hangtér egyszerűsített hullámegyenletét felírva és a megfelelő peremfeltételeket kielégítve, eredményül azt kapjuk, hogy mindkét sípban állóhullámok alakulnak ki, amelyek hullámhossza mindkét végén nyitott sípnál a következő képletből kapható meg:

$$l = (2k)\frac{\lambda}{4}, \qquad k = 1,2,3, \dots$$

Egyik végén zárt sípra felírt egyenlet pedig a következő:

$$l = (2k-1)\frac{\lambda}{4}, \qquad k = 1,2,3, \dots$$

,ahol a l a síp hossza, λ a kialakuló hang hullámhossza, k=1-et behelyettesítve kapott hullámhosszhoz tartozó frekvencia lesz az alapfrekvencia. [1]

Fontos megjegyzendő dolog, hogy a fenti egyenletekből két téves következtetés is levonható, miszerint azt feltételezhetnénk, hogy a sípoknak végtelen harmonikus komponense van, valamint ha mindkét végén nyitott síp egyik végét lezárjuk, akkor a keletkező hang hullámhossza l/2-ről l-re változik, ami a hang frekvencia feleződését (1 oktávval való csökkenését) eredményezi. A téves feltételezések oka az, hogy mindkét következtetés csak egydimenziós sípokra érvényes, vagyis nem veszik figyelembe a rezonátor átmérőjét. Ezért egyrészt a létrejövő legnagyobb frekvenciájú harmonikus komponensre is teljesülnie kell, hogy $\lambda_{min} > d/4$ [2], valamint a véges átmérő miatt a nyitott végek peremfeltételei nem a síp végén lesznek, hanem „kitolódnak” a sípból. A kitolódás elméletére a fenti egyenletek a következőképpen alakulnak:

$$l + \Delta l_1(d) + \Delta l_2(d) = (2k)\frac{\lambda}{4}$$

mindkét végén nyitott sípra, illetve

$$l + \Delta l_1(d) = (2k-1)\frac{\lambda}{4}$$

mindkét végén lezárt sípra, ahol a Δl_1 jelenti az ajaknál fellépő korrekciót, Δl_2 pedig a nyitott végénél fellépőt. Ebből következik, hogy a fele akkora frekvenciájú síp hossza, nem lesz fele olyan hosszú, mint a nyitott, továbbá két azonos alapfrekvenciájú eltérő *menzú*rájú sípok sem lesznek azonos hosszúságúak (a d átmérő miatt).

A sípok hangjának keletkezéséhez és annak helyes szintéziséhez szükséges megemlíteni, hogy a sípban keletkező hang hogyan csatolódik ki a külvilág felé. Az ajaksípoknál a nyíláson keresztül és a rezonátor falain át történik. Azonban bebizonyosodott,

hogy a falak rezgésének hatásfoka elhanyagolható, ezért modellezéskor a mindkét végén nyitott sípokat két pontszerű sugárzóként, a fedetteket pedig „ajaksugárzóként” kell értelmezni.

1.3. A nyelvsípok

A nyelvsípok erőteljesebb hangzásúak az ajaksípoknál. Hangkeltésük rezgőnyelves jellegű, tehát a beáramló levegő rezgésbe hozza a testben lévő fémnyelvet. Ez az elv a klarinét hangszerével megegyező mechanikájú, mert a rezonátor már nem tudja nagy mértékben befolyásolni a megszólaló hangot, a rezgőnyelv „dominál”. A tölcsér a hang kicsatolásának a mechanizmusába avatkozik bele, a spektrumát befolyásolja, ugyanis jóval nagyobb felharmonikustartalommal rendelkeznek a nyelvsípok az ajaksípoknál. Alkalmazásuk ritkább az erőteljes hang miatt.

A hang külvilágra sugárzása (kicsatolása) a síp nyitott végén jön létre.

1.4. Az orgona hangjának pszichoakusztikai jellemzői

Ha egy gitáros hangszert választ magának, az üzletben kipróbálja, értékeli, és ha jónak találja megveszi. Ez a jelenség zenészkörökben általános érvényű. Hiába ismerjük a hangszerünk paramétereit vagy anyagát (elektromos gitár esetében például mocsári kőris, mahagóni vagy éger test, ragasztott vagy csavarozott juhar nyak, esetleg ében fogólap, juhar nyakpickupok[3], három vagy kétállású kapcsoló), a hangját csak akkor ismerjük meg, ha „találkozunk” vele, kipróbáljuk és értékeljük. Egy hangszer hangját számtalan tényező befolyásolja. Húros hangszereknél a húrok romló minőségi állapota, elkopása az eredetihez képest megváltozott hangzást eredményez. Egy új húrokkal felszerelt gitár „kattogósabban” szól, míg egy elnyűtt, lecserélendő húrkészlet tompább hangzást generál, az említett magas komponensek eltűnnek, unalmasabb lesz a hangszer hangja. Egy adott hangszerkészítő mester vagy vállalat nyújthat olyan minőségű hangszereket, amelyek minőségben, hangzásban azonosítják őt.

Fontos hangzást befolyásoló tényezőnek nevezhető az is, hogy az adott terem teremakusztikai szempontból hogyan adja vissza a hangszerünk hangját. Nem mindegy, hogy hol helyezkedünk el a hangszerünkkel, valamint az is lényeges, hogy milyen reflexiós tényezőkkel rendelkeznek a körülöttünk lévő falak. Típus alapján is (és természetesen ára szerint) képesek lehetünk kategorizálni minőségi szempontból hangszereket, de a hallgatóság felől érkező szubjektivitással mindig számolni kell. Minden zenésznek kialakul a saját ízlése, amit jobban kedvel és jobb minőségűnek könyvel el, de vélemények kutatásakor el kell különítenünk a „vájtfülű” hallgatóságot (gondoljunk csak egy orgonistára, aki pontosan fel tudja ismerni a barokk, romantikus, és modern orgonák közötti különbségeket, és tudja a hangszer karakterében rejlő különbségek okát), az egyszerű műkedvelőktől

(akik adott esetben kedvelhetik az éppen hallható orgona hangját, de nem tudják megfogalmazni miért). [1] A Hammond-orgonának a jellegzetessége volt az állandóan forgó szinuszgenerátorok miatti pukkanás, mert általában nem a szinuszgenerátorok nullátmeneténél lett a hang leütve. Ez tette „Hammondossá” az elektronikus orgona hangját, ami egy mérnöki pontatlan megvalósításból fakadt, de később a hangszer egyedi megváltoztathatatlan

[3] Pickupoknak a gitár hangszedőit hívják zenész körökben.

jellegének könyvelték el az orgonisták.

A pszichoakusztika ezen hangérzetek leírására, egy adott jelenség objektív, és fizikai jellemzőjének kapcsolatának leírására törekszik, tehát a hangérzeti különbségeket fizikai magyarázatokkal támasztja alá, valamint kutatja a minél jobb megoldásokat. Ezek a kutatások mérések formájában történnek. Egy adott hallgatóságot vetnek alá adott hangérzetű jelenségnek ugyanolyan környezeti feltételek mellett, és a hallgatóság minden tagjának független, objektív döntést kell hoznia az adott jelenségről, miközben annak fizikai paramétereit mérik.[1]

A következőkben a zenei hangok fizikai jellemzőinek bemutatása, csoportosítása következik, rátérve az orgona hangjának analízise által meghatározott, orgona hangszintézishez szorosan kapcsolódó tulajdonságaira is.

1.4.1. Az állandósult spektrum

A hangszerek akusztikailag több szempontból csoportosíthatók. A szintézis szempontjából egyik legfontosabb a gerjesztés szerinti csoportosítás. Gerjesztés szerint kétféle típust különböztethetünk meg: az egyik az impulzusszerű energiabevitelű hangszerek csoportja, a másik a folyamatos energiabevitellel gerjeszthető hangszerek.[5] Az első típusú hangszereknél egy impulzuslöket hozza létre a hangot, amely közelítően a gerjesztés pillantában eléri a maximális amplitúdóját, majd egy folyamatos exponenciális jellegű lecsengést produkál (ilyen hangszer például a dob, zongora, xilofon és az összes pengetett hangszer is mint a gitár, a bendzsó, a citera, stb..). A második típusban a gerjesztés folyamatosnak tekinthető időben, így a hangszer képes elérni az állandósult állapotot. Ezt a kategóriát képviselik a vonós (hegedű, brácsa) és a fúvós (fuvola, oboa, orgona) hangszerek.

Állandósult spektrumról csak állandósult gerjesztésű hangszerek esetén beszélhetünk, amely a hangszer hangja frekvenciakomponenseinek amplitúdóját ábrázolja a frekvencia függvényében. A spektrumból kinyerhető információk között a legfontosabb talán a hang alapharmonikusa, és annak frekvenciája. Ez a hangmagasság hangérzeti jellemzőt befolyásolja, a felharmonikusok pedig a hangszínért lesznek felelősek attól függően, hogy az mekkora és milyen számú felharmonikus komponenseket tartalmaz. Ezeken kívül számos esetben megtapasztalhatóak a harmonikusok helyétől eltérő komponensek is, amelyekre az emberi hallás nagyon érzékeny. Ez a jelenség a gerjesztőjel és a hozzá kapcsolt rezonátorok közötti felhang-struktúrájának a különbségéből adódhat, de a zajjelenségek is ide sorolhatók, mint a harmonikusoktól való eltérő, kiemelkedő komponensek.

Az orgona analízisét tárgyaló dolgozat leírja, hogy adott síp spektrumának felharmonikustartalma erősen változik az anyaga, az alapharmonikusának frekvenciája függvényében, valamint attól is függ, hogy milyen a síp típusa (fedett, nyitott). Érdekesség például, hogy két különböző mester által készített megegyező anyagú és menzúrájú sípok felharmonikustartalmának számában is előfordulhatnak jelentős különbségek. [1] A mérések megmutatták, hogy például egy Principál síp harmonikusainak a száma a néhánytól[4] (három, négy), egészen a több tucatig (a húsz harmonikusnál többig) is változhat, valamint a fémsípok több, a fasípok kevesebb harmonikust tartalmaznak[4]. Általánosságban is elmondható, hogy a mélyebb sípok nagyságrendekkel több harmonikust is tartalmaznak a magasabbakénál. Az is

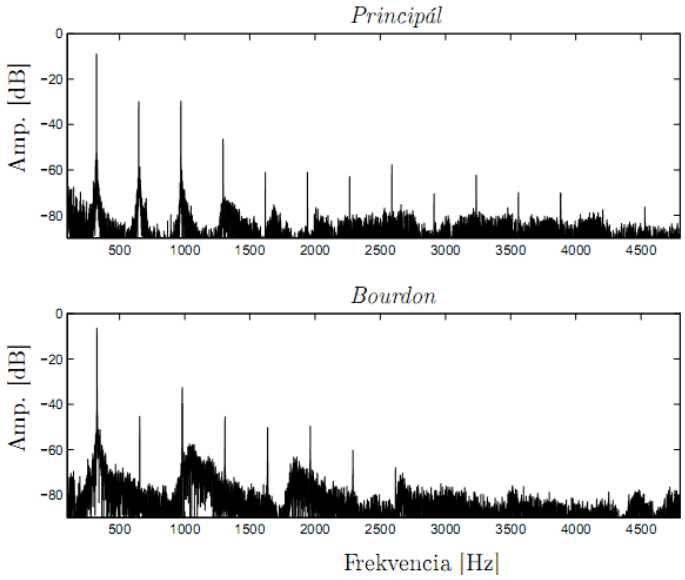
[4] *A szintetizátor plugin a harmonikusok számát a megadott paraméterek alapján 3- 25-ig tudja kezelni*

bebizonyosodott, hogy a fedett sípokat tartalmazó regiszterek melletti mérések spektrumképein a páratlan felharmonikusok amplitúdói határozottan kisebbek, mint a nyitott sípoké. Ennek a fizikai magyarázata a következő:

$$f_{ny,k} = \frac{c}{\lambda_n} = 2k\frac{c}{4l} = k\frac{c}{\lambda_0}$$

$$f_{z,k} = \frac{c}{\lambda_n} = (2k-1)\frac{c}{4l} = (2k-1)\frac{c}{\lambda_0}$$

Az első képlet a nyitott sípokban kialakulható lehetséges frekvenciákat mutatja, a második pedig a fedett sípokra vonatkozik, ahol λ_0 a hang alapharmonikusának hullámhossza [1], c a hang terjedési sebessége levegőben, l pedig a síp hossza. Ebből következik, hogy ideális fedett sípoknál nem alakulhatnának ki páros felharmonikusok a síp sektrumában, a valóságban természetesen nem létezik ideális lezárás sípoknál, de a jelenség „mérsékelten" fennáll, ami a páros harmonikusok lecsökkent amplitúdóiban mutatkozik meg. Az ábrán is megfigyelhető nem harmonikus komponensekhez tartozó zaj, és ehhez kapcsolódóan az egyes harmonikusokkal megegyező helyen kialakult zajcsúcsok is, melynek ismertetése a következő, sztochasztikus jelenségek fejezetnek a témaköre. [1]



2. ábra: Egy nyitott Principál és egy fedett Bourdon síp spektruma

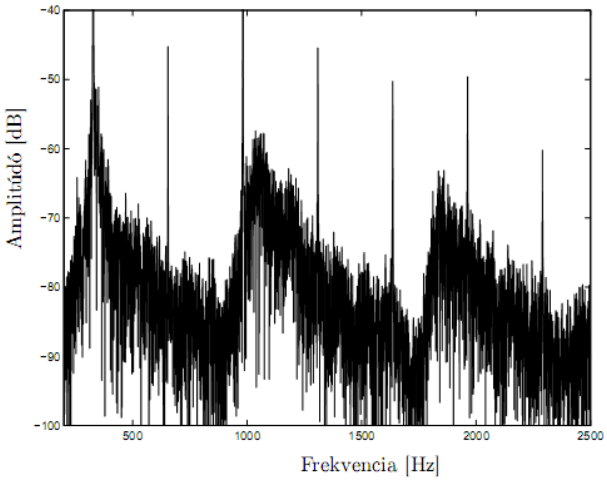
1.4.2. Sztochasztikus jelenségek

Egy állandó gerjesztésű hangszer sípjának hangja minden esetben tartalmaz a spektrumban is észlelhető nem periodikus komponenseket. Ezek a nem periodikus jelek bizonyos esetekben hallhatóak is. A nem periodikusság egyik oka a tranziens jelenségekben keresendő, amelyek elmúlásával állandósult állapotot alakul ki. Viszont állandósult állapotban is észlelhető a non-periodicitás, amelyért a hangszer hangjának általános zajjelensége tehető felelőssé. A zaj, mint sztochasztikus jelenség nagyban hozzájárul a hangszer hangjának valódiságához,

színezéséhez, ami orgonák esetében hallhatóan karakteres hangot kölcsönöz a hangszernek. A sípok sztochasztikus jellemzői tehát elsősorban a gerjesztés (levegőnyelv) statisztikus ingadozására és a gerjesztés-rezonátor kölcsönhatás instabilitására vezethetők vissza. Ezek a hangkeltéskor mérhető instabilitások nemcsak zajként jelentkezhet, hanem az egyes harmonikusok amplitúdó- vagy frekvenciamodulációját, vagy lassú fázisvándorlását is okozzák.[2]

A harmonikusok tárgyalásánál szó volt az emberi hallás érzékenységéről olyan jelenségekre, amelyek nem a harmonikus komponensekkel vannak összefüggésben. Az emberi fülre tehát pszichoakusztikai szempontból nagy hatással van a szélessávú zaj, vagy a frekvencia lassú változása, mert hallószervünk fontos tulajdonsága, hogy a jel változásaira kifejezetten „fogékony".

Orgonáknál a megszólaltatott hang vezérlés szempontjából nem változtatható, ezért az orgonasípokra jellemző sípzaj nem függ magától a vezérléstől, csak a gerjesztést végző levegő mozgásának véletlenszerű (statisztikus) ingadozásától, amely instabilitás forrása a rendszerben. A következő ábrán egy Bourdon típusú síp zajspektruma látható.



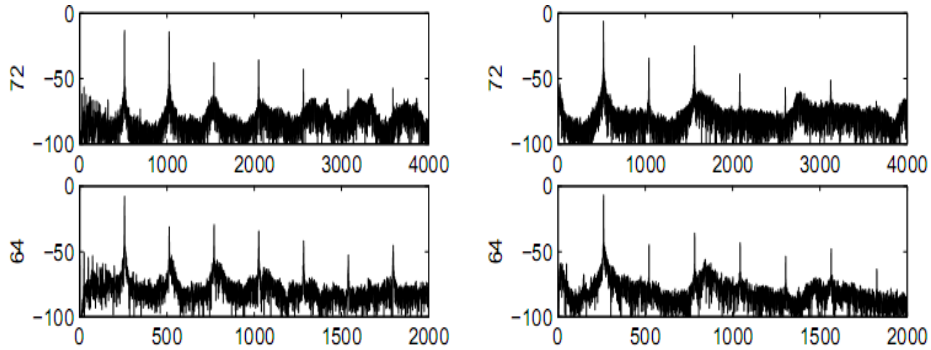
3. ábra: Orgonahang spektruma

Az ábrán jól megfigyelhetőek a korábban említett zajcsúcsok, amelyek bizonyos esetekben kapcsolódnak a harmonikus csúcsokkal (az ábrán az alapharmonikuséval), tehát a egyes harmonikus komponensek közvetlenül zajcsúcsból emelkedik ki. Ebből kifolyólag megkülönböztethetünk a harmonikusokra illeszkedő, valamint nem illeszkedő zajcsúcsokat (amelyet a szakirodalom szinkron és független zajcsúcsoknak hív). A szakirodalmak alapján a hangszer spektrumbeli zajcsúcsok keletkezéséért az ékhang-generátorok a felelősek.

Korábbi dolgozatok alapján zajcsúcsok keletkezésére két fő elmélet létezik. [1] Mindkét elmélet a zajcsúcsok frekvenciafüggését hivatottak bebizonyítani. Az egyik elmélet alapján a hangszernek az ékhang a bemenőjele, amelynek hatására a rezonátor megszólal, és így alakulnak ki a felharmonikusok.[4] Más mérések szerint ezek a zajcsúcsok nem függetlenek a harmonikusoktól, hanem dinamikusan alakulnak ki. Ezek szerint önmagában a zajgenerátor-rész frekvenciafüggetlen zajcsúcsokat generálna, de a rezonátor a síphang keltése után rákényszeríti rezgését a generátorra, és ezen visszacsatolás után alakul ki az eredő spektrum.

A zaj változása az alapharmonikusok változásával is megfigyelhető volt az analízis során. Ezek alapján kétféle regiszterről beszélhetünk. Az első típusba tartoznak azok a

sípsorok, amelyek az azonos hangzásérzet kialakítása miatt paraméterek pontos beállításával biztosították a sípok hasonló zajgenerálását, vagyis minden síp hasonló, karakteres zajt generált adott regiszteren belül. A másikba tartoznak azon regiszterek, amelynek sípzajai eltérő jellegűek. Ez azt jelenti, hogy a harmonikusok és a zajcsúcsok egymáshoz való viszonya (aránya, helyzete) nem egyező, és a zajcsúcsok formája is nagyban eltér. Az ábrán Principál és Bourdon regiszterhez tartozó sípjain felfedezhetőek az azonos típusú zajcsúcsok hasonlóságai.



4. ábra: Principál és Bourdon síp spektrumai

Összességében elmondható, hogy gyakorlatban is tapasztalt tény, hogy a zajcsúcsok adják a sípok egyedi, megkülönböztethető „orgonás” hangzását.

1.4.3. Tranziens jelenségek

A zenei hangok nulla jelszintű állapotból a megszólalási állapotig történő eljutásához egy nagyon rövid ideig tartó (orgonáknál ez körülbelül a másodperc negyede, tizede nagyságrendbe eső) felépülő állapoton keresztül jut el. Ezt állandó gerjesztésű hangszereknél tranziens jelenségeknek hívjuk. A tranziens jelenségekbe azonban nemcsak a hangszer berezgése, hanem az állandósult állapotból nulla állapotig történő lecsengését is értjük. Az impulzusszerű hangszerek – mivel nem rendelkeznek állandósult állapottal – az egész hangjelenség tranziensnek tekintendő. Ezt úgy tudjuk elképzelni, hogy egy megpengetett gitár nagyon rövid idő alatt eléri amplitúdójának csúcsát, és a felépülő tranzienshez képest jóval hosszabb ideig tartó lecsengés jelenti a hangesemény nagyobb részét. A tranziens jelenségek fontosságát bizonyították azokkal a kísérletekkel, amelyek hallgatóságot olyan hangszerek (gitár, csemballó) hangjának tették ki, amelyek elejéről levágták a tranziens jelenségeket, és ebből következően azonosíthatatlanná váltak az egyébként könnyen felismerhető hangszerek.[4]

Minden állandó gerjesztésű hangszernek ismert a jelszerkezete (ahogyan az orgona hangjának is), ami spektrumkép formájában jelenik meg. A spektrumkép vonalas szerkezetű, vagyis beszélhetünk harmonikusokról, zajkomponensekről, valamint néhány nem harmonikus komponensről is. A spektrumkép feltételezhetően nemcsak állandósult állapotban, hanem a tranziens berezgési és lecsengési ideje alatt is ilyen formájú marad. A feltételezés azért szükséges, mert a tranziensek több lépéses spektrum ábrázolása az ismert jelfeldolgozási

módszerek szerint fizikai korlátokba ütközik, többnyire pontatlan ábrázolásai lehetségesek[5].

Korábbi analízisek, valamint orgonairodalmak alapján elmondható tehát, hogy az egyes harmonikusok és a maradék nem harmonikus komponensek tranziens leírás szempontjából külön tárgyalhatók, amely lineáris megközelítést jelent.[1] Azért nevezhető a vizsgálat közelítően lineárisnak, mert a korábbi vizsgálatok megmutatták, hogy a tranziens alatt csak a sípzaj, és a harmonikusok aránya változik, illetve a sípzajon belül az amplitúdórányok módosulnak, a frekvenciakomponensek megközelítőleg állandónak mondhatók. A komponensek tehát joggal, független vizsgálhatók. Az ilyen vizsgálatokat könnyen alátámaszthatják a valóságban tapasztaltak. Orgona szakirodalmak ugyanis kifejtik, hogy egyes orgonisták és orgonaépítők szerint azok a sípok nevezhető minőséginek, amelyek megszólalását egy rövid zajjelenség kíséri, majd mindegyik komponens egyenletesen éri el a maximális amplitúdóját.[4] Más vélemények szerint pedig a síp akkor jó, ha az elején egy magasabb frekvencián szólal meg, majd az energia fokozatosan áramlik az alapharmonikusba. Ez a magasabb frekvencia az harmonikus frekvenciáktól eltérő általában. Orgonairodalmak az utóbbi jelenséget „köpködésnek” is hívják.

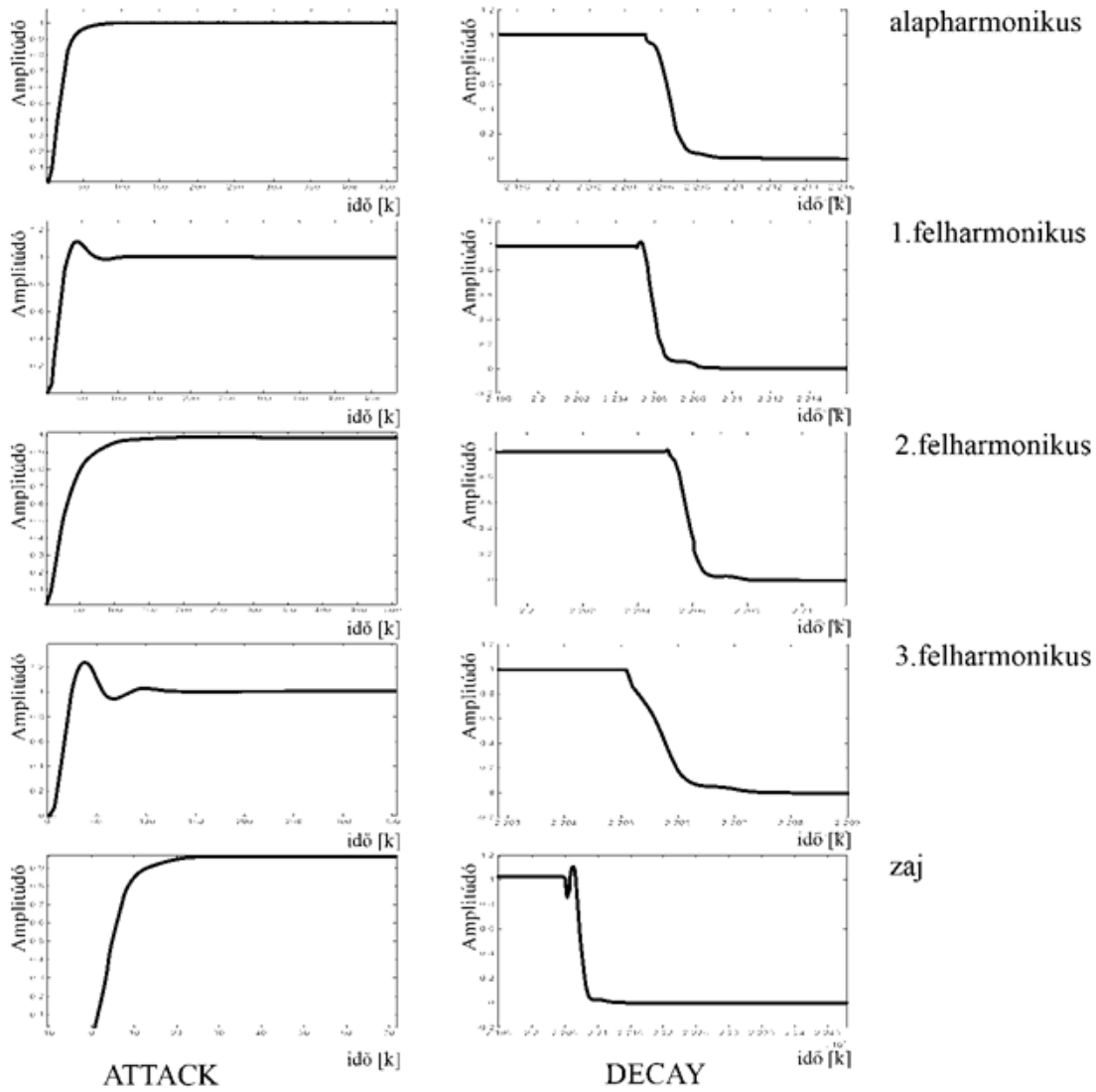
Fizikai leírás szempontjából a tranziensek kialakulásának okai a következők: A sípba beáramló levegő nekiütközik a felső ajaknak, ami oszcillációt hoz létre, ezáltal zenei hang keletkezik. Ezt hívják ékhangnak. Az előző – stochasztikus jelenségeket tárgyaló – fejezetben is leírva az ékhang által keltett rezgés felelős a kialakult zajcsúcsok keletkezéséért, amely gerjeszti a síptestet (rezonátort). A rezonátor válasza lesz a harmonikusok kialakulása. A tranzienseknek aszerint létezik több típusa, hogy a harmonikus és a hozzá tartozó zajcsúcs milyen viszonyban áll egymással, ugyanis amelyik harmonikus közelebb van a zajcsúcshoz hamarabb berezeg, így az adott hang hamarabb eléri amplitúdójának csúcsát. Amennyiben a gerjesztőjel kiemelt frekvenciái viszonylag jobban eltérnek a cső harmonikus frekvenciáitól, a megszólalás folyamata hosszabb lesz, a hang nehezebben rezeg be[6]. A köpködés pedig úgy jön létre, hogy az ékhang gerjesztés zajcsúcsa valamely harmonikushoz közel helyezkedik el, berezeg, majd fokozatosan beáll az adott harmonikus frekvenciájába.[6]

A tranziensjellemzők nemcsak síptípusonként lehetnek eltérőek, hanem jellegzetes képük és felfutási idejük az adott síp hangmagasságának is függvénye. A hangszerekre általában jellemző, hogy a magasabb frekvenciájú hangok hamarabb érik el az állandósult állapotot, ami az orgonáknál is fennáll (A mély hangok „lomhaságát” a szintetizátor modellje is jól szemlélteti). [6]

Végeredményben elmondható, hogy a harmonikusonkénti vizsgálat a 2.2.2. fejezetben ismertetett jelmodell alapú koncepcióhoz jól illeszkedik. A tranziens-burkolók megvalósításának lépéseit is ez a fejezet fejti ki. A következő ábrán látható egy Principál síp néhány komponenseinek tranziens-burkolói, a hang berezgését és lecsengését egyaránt szemléltetve.

[5] A pontosság növelésére ugyan létezik néhány eljárás, név szerint spektrogramm készítés átlapolódó futóablakos FFT alkalmazásával, Wigner-disztribúció, Cummulative Attack és Decay Spektrum meghatározására szolgáló eljárások, de ismertetéseikre részleteiben nem tér ki a dolgozat.[1]

[6] A barokk korban készült orgonák lassabb megszólalásúak ellentétben a romantikus orgonákkal, ahol sípok a gyors megszólalásra lettek beállítva. Ez a fő jelenség, ami miatt az orgonisták meg tudják határozni a gyártás időpontját.



5. ábra: A tranziens-jelenségek szemléltetése időtartományban

1.4.4. Külső körülmények

A hangszerek hangjának pszichoakusztikai jellemzőihez szervesen hozzátartozik a külső körülmények figyelembevétele. Az egyik ilyen jelenség a csatolás, ami csak akkor észlelhető bizonyos hangszereknél, ha egyszerre több hang szólal meg.

Ennek az egyik fő oka, hogy például zongoránál az egyik megszólaló hang a közös sugárzófelületen megrezegteti a felhangjainak megfelelő húrokat, amennyiben a zengetőpedál ezt engedi. Az orgonánál nem ez a csatolás jellemző inkább – mert ott minden hang megfelelő paraméterű síppal rendelkezik – hanem a megszólaló síp és felhangsípjai[7] között olyan kapcsolat jön létre, amely nemcsak a hangerő megnövelését okozza, hanem teltebbé „vaskosabbá” teszi a hangját [2]. A jelenség oka a fent említett sípok közötti interferenciák,

[7] Felhangsípnak nevezik azt a sípot, amely egy adott billentyűhöz tartozik, de alapfrekvenciája n-szerese a billentyűhöz tartozó hangénak

lebegések, amelyek kórusszerű hangot kölcsönöznek a hangszernek.

Másik külső körülmény, amiről érdemes szót ejtenünk a Mitnahme-effektus[1]. A jelenség nagyon érdekes, mert az azonos névleges frekvenciájú sípok spektrumbeli eltéréseinek a kiegyenlítéséért felelős. Ha ugyanis két elvileg azonos hangmagasságú síp keltette hang között apró eltérés van, akkor az egyik hang rákényszeríti a rezgését a másikra, kiegyenlítve a két síp valós frekvenciája közötti különbséget. Más néven „magával vivő hatásnak" is nevezik a jelenséget. A hamis hang bizonyos korlátok között tehát kitisztul.

A harmadik hatás az orgona regiszterei és sípok közötti térbeli eltérésből fakad. A hangszer nagy mérete miatt a különböző sípokból a hallgatóig eljutó hang hosszabb utat tesz meg, mint a másik és ez a távolság nem elhanyagolható a hallgató hangszertől való távolságához képest. Ez a távolságkülönbség szintéziskor a különböző regiszterek sípjai közötti hangerőkülönbséggel jól modellezhető. Ez természetesen sztereó, vagy akár többcsatornás rendszert is jelenthet, attól függően, hogy mekkora költséggel akarjuk ezt létrehozni.

Összességében is elmondható, hogy a terem akusztikai tényezői is nagyban befolyásolják az orgona hangjának értékítéletét. A jobb akusztikai tulajdonságokkal bíró templomterem képes egy gyengébb orgona hangjából többet kihozni, mintha egy akusztikailag kevésbé előnyös, de „minőségibb" hangszert hallgatnánk.

Egyéb orgona hangját befolyásoló tényezők közül – amit a zenei szakértők, orgonakészítők a hangszer hangjában azonosítani tudnak – a levegőt tartalmazó szélládák sziszegése, vagy a fujtatómotor zúgása. Sokszor a megszokott jellegzetes zaj hiánya kelt a szemlélőben olyan érzést, ami miatt nem tartja elég „sípszerűnek" a hangszer hangját, vagy a 2.1.1. fejezetben bemutatott Hammond-orgona pukkanásszerű megszólalását tulajdonították a hangszer elidegeníthetetlen jellemzőinek.

A félév során végrehajtott jelmodell alapú szintézis nem veszi figyelembe a külső körülmények megvalósítását, de a modell kitűnő továbbfejlesztési felületét képezi.

2. Orgonahang szintézise

A hangszintézis[8], vagy szintetizálás általános érvényű kifejezés. A szintézis fogalmán egyedi, jól megkülönböztethető összetevők megfelelő összegzését értjük egy adott cél érdekében. Mivel a szintetizátorok eredetileg akusztikus hangszerek hangjának utánzására készültek – és csak később lett a fantázia és kreativitás kifejezőeszköze, új hangzások tárháza – a szintetizálni kívánt hangszer hangját összetevőkre kellett bontani, analízisnek vetették alá. Célra vezető módszer, ha a hullámalak spektrumát analizáljuk. A fejezetre előretekintvén tehát megállapítható, hogy a különböző szintézisre alkalmas eszközök és módszerek feladata ezeket a komponenseket, mint harmonikusokat precíz oszcillátorokkal, tranziens jelenségeket a megfelelő időállandókkal, a zajjelenségeket pedig szubtraktív módon a megfelelő paraméterű szűrők segítségével előállítani.

2.1. Orgonaszintézis módszerek

Az alábbiakban az orgona legáltalánosabb szintézis módszerei kerülnek bemutatásra, konkrét megvalósított szintetizátorokon keresztül, időben előrehaladva.

2.1.1. Additív szintézis

Az additív szintézis a periodikus jel Fourier-sorral való közelítésével van kapcsolatban, miszerint minden végtelen spektrumú periodikus jel előállítható nem véges számú, különböző amplitúdójú szinuszos jellel, sávkorlátozott esetre szorítkozva pedig véges számú szinuszos jellel[10]. Az additív-szintézist Fourier-szintézisnek is nevezzük. Az első egyszerű szinusz oszcillátorok hangjának összeadása elvén működő egyik korai jelentős szabadalom a Hammond-orgona volt. Célja az orgona hangjának élethű reprodukálása volt. Az eszközt Laurens Hammond hozta létre az Egyesült Államokban 1934-ben.

A működés elektromechanikus elven szolgáltatott különböző frekvenciájú szinuszjeleket (91-et), amelyek a forgási sebességgel voltak állíthatóak. [17]Így változó mágneses mező keletkezett, amiket terkercsek segítségével alakítottak át elektromos jellé.
A vezérlést szolgáló megfelelő billentyűkhöz a szinuszjelek a megfelelő frekvencia szerint 9 harmonikus szólaltatott meg. Ezeket szorozva a megfelelő amplitúdókkal különböző hangszínek jöttek létre. Az így előállított harmonikusokat egy hangfrekvenciás transzformátor összegezte, előállítva a kimeneti jelet.

Az orgona hangját olyan beépített effektek alkalmazásával színesítették, mint például zengető, kórus vagy visszhang, később az 50-es évek közepén pedig a tranziens modellezést módosító *percussion* egységet is tettek az orgonába. [1]

A Hammond-orgona a maga korában nagy mérnöki vívmánynak számított, de inkább a könnyűzenében terjedt el, mert a hangja nem produkált sípszerű hangzást. Ennek egyik oka az volt, hogy a valóságban a felharmonikusok száma az adott alaphangtól erősen függ, és ezt a

[8] A szintézis görög eredetű szó, összegzést, kapcsolást, összeállítást jelent.

modell nem adta jól vissza. Másik említésre méltó tulajdonsága pedig, hogy a kitartott hang hangszíne megváltozik más hangok lenyomásánál. A szinuszgenerátorok nem feltétlenül nulla kiinduló fázissal rendelkeztek állandó működésük miatt, ami pukkanást okozott a hang lenyomásánál (tehát hiányzott a burkológörbék által előállított tranziens a rendszerből), valamint az effektek nem biztosították az orgonás hangzást. Ezek inkább egyedi hatást keltettek, mint az orgona hangjához közelítettek volna.

Összességében nagy sikerű additív szintézisen alapuló hangszert gyártottak, de a szintézis eljárásnak egyik nagy hátránya, hogy a nagyszámú felharmonikus (és burkológörbe-generátorok) előállítása abban a megfelelő digitális kapacitásokat nélkülöző korszakban ezekkel a módszerekkel meglehetősen bonyolult és költséges volt.

2.1.2. Szubtraktív szintézis

Éppen a nagyszámú felharmonikusok előállításának igénye miatt jöttek létre a Hammond-orgona hatására olyan orgona hangját modellező analóg hangszerek, amelyek szubtraktív szintézisen alapulnak. A szubtraktív eljárás lényege, hogy a jelgenerátorunk egy harmonikusokban gazdag jelet állít elő, mint például a négyszög-, fűrész,- vagy háromszögjel, és megfelelő (LFO-val állítható töréspontí frekvenciájú) szűrőkkel kialakítja a kívánt spektrumot, aminek előállítása viszonylag olcsón végrehajtható.[19] Az oszcillátorok frekvenciája és amplitúdója is állítható. Az ilyen jelek mellett tipikusan zajgenerátor által előállított fehér vagy színes zajt is alkalmaznak, hogy a jellel együtt kialakítsák a kívánt spektrumot. Szubtraktív eljárást használunk, amikor az orgona hangjának szintéziséhez állítjuk elő a megfelelő additív zajt, amelynek felharmonikusokban gazdag bemenőjele Gauss-zaj lesz.

Az analóg áramkörös szubtraktív elven működő orgonák a dús felharmonikusokkal teli hangokat nagyon jó közelítéssel előállítják, de a kevesebb felharmonikusszámú hangokat nehézségekkel lehet csak megvalósítani. Állandósult állapotban ezek az orgonák ritkán tartalmaznak zajt, inkább modulációkkal modellezik a sztochasztikus jelenséget. Ez azokra a harmonikusokra, amelynek amplitúdói nagyságrendekkel eltérnek a zaj nagyságától, nem indokolt, így a hangzásban is eltérés tapasztalható az eredeti hangszer és a modell között.

Nagyon nagy hátránynak könyvelhető el a tranziens implementálásának magas költsége, ugyanis a modell alapvetően nem támogatja a harmonikusonkénti amplitúdó változás implementálását. Ennek ellenére vannak létező modellek, amelyek köpködő megszólalás problémáját megoldották. A teremérzet-hatás kialakítása a modellben nem volt jellemző, ugyanis nagy termekben használták őket megfelelően erősítve, ahol a teremérzet a valóságban is létrejött.

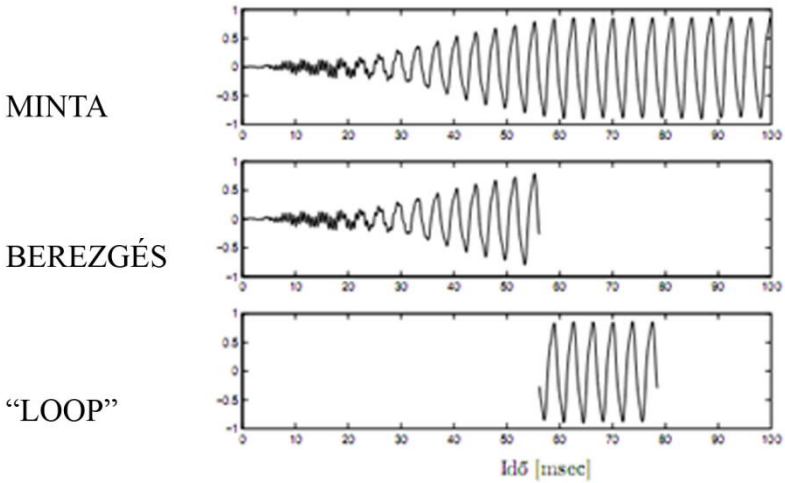
Az első szubtraktív szintetizátort Robert Moog 1964-ben készítette el.

2.1.3. Hullámtáblás szintézis

A hullámtáblás szintézis előretörése a digitális technika fejlődésének volt köszönhető. Különböző hangszerek szintézisénél ugyanis felvetődik sokakban a kérdés, hogy miért ne modellezhetnénk úgy a hangszer hangját, hogy optimális körülmények között rögzítjük a különböző hangokat, és azt a megalkotott szintetizátorunk megfelelő lenyomott billentyűjénél visszajátsszuk. Ezen az elméleten alapul a mintavételezéses eljárás, amelyet PCM-nek, valamint „Sampler”-nek is szoktak nevezni. Az eljárás ameddig a felvett minták

minőségromlásnak voltak kitéve (analóg tárolási módszereknek köszönhetően), nem volt túl elterjedt [9]. A minták tárolásának a helykapacitás véges korlátai szabtak gátat, egyúttal gátat szabva a hangszer minőségének is, ugyanis ha visszajátszáskor egy nagyon rövid mintát ismételtek folyamatosan, jelentősen le tudták csökkenteni a hang hűségét az eredetihez képest. A túl hosszú hurkok tárolása pedig fizikai korlátokba ütközött, ennek ellenére előnyként lehetett elkönyvelni, hogy ezzel a módszerrel korlátlan gazdagságú hangszín volt előállítható.

A teljességhez hozzátartozik, hogy csak az állandósult állapotbeli mintát kellett ismételni, a tranziens modellezést tartalmazó mintát pedig megfelelően vezérelve a lejátszás elején és a végén lehetett illeszteni a hanghoz. Az 4. ábra szemlélteti a kezdeti amplitúdóváltozás szakaszát, majd az állandósult állapotú hurkot.



6. ábra: A jel berezgési szakasza és állandósult állapota

A modell hanghűsége azonban így sem volt tökéletes, ugyanis nem egy adott hangszer hangját tárolták el, hanem a felvétel ideje alatt lévő hangját, tehát csak azt tükrözte, hogy egy adott pillanatban és egy adott távolságban hogyan szólalt meg a hangszer. A módszer sztochasztikus jelenségek modellezésére teljesen alkalmatlan, mert a lejátszott hangunk determinisztikus és periodikus. Nagy PCM-szintézis elven működő orgona szintetizátor cégek (mint a Rodgers és Albhorn) külön implementálják a modellbe a zajt, ami a PCM-ből való kilépést jelenti. Tranziensek modellezésénél is a PCM egy mintát tárol, aminek állandó visszajátszása szintén „szegényessé” teszi az eszközt, mert a hang felépülése és lecsengése a sípokban is gyakorlatilag sok szempontból sztochasztikus.[1]

2.1.4. Fizikai-modell alapú szintézis

A fizikai-modell alapú szintézis a fentiektől a hangszer megközelítésében tér el jelentősen. Az elv ugyanis a hangszer fizikáját hívja segítségül, pontosabban a hang keletkezésének a fizikai

[9] Ennek ellenére stúdiókban bizonyos ideig alkalmazták. A 60-as években népszerű Beatles popzenekar által készített felvételeken is feltűnt PCM elvű orgona. Az egyes billentyűkhöz szalagokon felvett hagminták voltak rögzítve, de csak a hang néhány másodpercnyi kitartására voltak optimalizálva.

hátterét használja fel arra, hogy egy nagyon precíz modellt kapjon. A fizikai szintézis lényege, hogy a hangszert mint rezgő rendszert modellezi.

Akusztikus hangszerek hangkeltési modellje megkülönbözteti a gerjesztőt, rezonátort, valamint a hangszer testét.[8] A gerjesztő-rezonátor kölcsönös viszonyáról már szó volt a tranziens és stochasztikus jelenségek leírásánál egyaránt. Orgonáknál a sípok paramétereinek felhasználásával áramlástanban ismert Navier-Stokes differenciálegyenlet-rendszer végeselem-módszerrel történő megoldásával megkapjuk az akusztikai tér minden pontjában a hangnyomás, a részecskesebesség időfüggését a beáramló sebességre, ami a sípfal, a magrés, az ajkak és a sípláb segítségével adható meg.[1] A kapott eredmények nagy pontossággal modellezték a levegő áramlását, de sok, modell szempontjából felesleges információt eredményeztek, ezáltal a megvalósítás költségét jelentősen növelték.

Fizikai alapú modellel a fentiekhez hasonlóan bármilyen hangszer modellezhető, azonban a modell használata egyszerűsítésekre kényszeríti a szintézist végző egyént. Az egyszerűsítés mértéke több szempontot is figyelembe vehet, de ha a célunk olyan módszer kidolgozása, amellyel a minél tökéletesebben reprodukált hangzást céloztuk meg, akkor a hang keltését nem befolyásoló tényezők elhagyása lesz az optimális megoldás.

Összességében a fizikai-modell alapú szintézis gyakorlati megvalósítása valós időben a magas költségek miatt akadályokba ütközik, de ha fenti modelleket tekintjük mindegyik rendelkezik valamilyen előnyös és hátrányos tulajdonsággal. Valójában a PCM-szintézis kiegészítve sztochasztikus jellemzőkkel elég jó kompromisszumnak bizonyult, melyet több szintézissel foglalkozó cég implementált termékeiben.

2.2. Jelmodell alapú szintézis

Minden szintézisnél alapvető cél olyan hangmodell megalkotása, ami a hallgatóság számára kielégítően hasonlít az eredeti hangszer hangjához. A jelmodell alapú szintézis lényege, hogy az emberi fül számára hallható, és fontos dolgokat megtartsa, a kevésbé fontosakat elhanyagolja, ugyanis néhány kevésbé fontos dolog mellőzésével a számítási idő jelentősen lecsökkenthető. Gondoljunk csak arra, hogy egy adott orgona regiszter esetén nem feltétlenül szükséges a regiszter összes sípjának a felvételét elkészítenünk, majd ebből egyenként a paramétert kinyernünk. A szintézis során ugyanis számos meglévő hangfelvétel alapján a paraméterek frekvenciafüggő viselkedései alapján becsülhetőek a le nem mért, esetünkben magas frekvenciás hangok paraméterei. Ezzel a kérdéssel a harmadik fejezetben a konkrét megvalósításnál foglalkozunk.

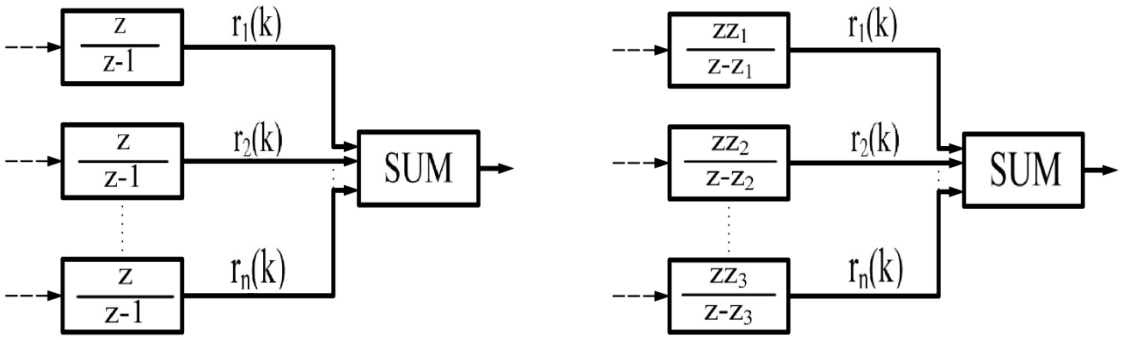
2.2.1. Koncepcionális jelmodell

Az orgona hangja és analízis fejezetnél láthattuk, hogy a sípok hangja döntően periodikus, és véges számú domináns diszkrét komponenst tartalmaz. Továbbá a tranziensek vizsgálatánál az is bebizonyosodott, hogy az egyes felharmonikusok egymástól függetlenül épülnek fel és frekvenciájuk a tranziensek közben nem változik. Kézenfekvő megoldásnak bizonyul, hogy az orgonahang koncepciója a periodikus jel diszkrét idejű modellje legyen.

2.2.1.1. A periodikus jel modellje

Fourier-sorfejtés által a jel felbontható szinuszos komponenseire. Ezt az elméletet

megfordíthatjuk, vagyis jelgenerátorok alkalmazásával szinuszos jelekből előállíthatunk bármilyen jelet.[1]



7. ábra: A periodikus jel diszkrét idejű modellje (a) időváriáns, (b) időinvariáns paraméterekkel

A két ábra jelgenerátorokat ábrázol, amelyek sávkorlátozott periodikus jelet állítanak elő N komponens összegeként. A jel úgy jön létre, hogy az ábrán látható N integrátor bemenetére egységimpulzust kapcsolva, a kimenetén megjelenő egységugrás k. értéke szorzódik a Fourier-sor megfelelő ún. exponenciális bázisfüggvényének k. értékével. Egy komponens bázisfüggvénye a következő:

$$r_i = e^{j2\pi k\frac{f_i}{f_s}}$$

A képletben f_s a rendszer mintavételi frekvenciája, f_i a bázisfüggvény frekvenciája, k a diszkrét időbeli értékek. A rendszerben lévő N darab csatornát összegezzük, és megkapjuk a modell kimenetét. [9] A kívánt jelalak előállításához a bemenetre kell kapcsolnunk a Fourier-sor komplex együtthatóit. Ezek az egyes jelkomponensek fázisai és amplitúdói lesznek. Ezeket kapcsoljuk az integrátor bemenetére.

Időinvariáns esetben modellünk a 7.ábrán lévő alakban szerepel. Az egyes komplex exponenciálisokat diszkrét idejű rezonátorok állítják elő. Egy f_i frekvenciát előállító komponens pólusának értéke:

$$z_i = e^{j2\pi\frac{f_i}{f_s}}$$

A komponens pólusa a megfelelő bázisfüggvény értékével kapcsolatban áll. Ahogy az előző modellben is, itt is a bemenetekre kell kapcsolnunk a megfelelő értékeket (amplitúdó, fázis) a $k = 0$ időpillanatban. Észrevehető, hogy a diszkrét idejű rendszer N pólust tartalmaz az egységkörön. Csak így lehetséges az, hogy a rendszer egy véges hosszúságú bemenőjelre (egységimpulzus) végtelen idejű választ adjon, vagyis a rendszer a stabilitás határhelyzetében működjön.

2.2.2. Orgonahang modellezés jelmodellel

A jelmodell segítségével előállítható egy végtelen idejű periodikus jel. A periodikus jelet a jelmodellel már nagyon egyszerűen előállíthatjuk. Az integrátorok bemenetét ellátjuk a megfelelő vezérlőparaméterekkel, majd a nem periodikus komponenseket arra alkalmas generátorokkal valósítjuk meg, amit a későbbiekben tárgyalt módon implementálunk a periodikus jelünkbe. Az egyes integrátorok kimenetei az egyes harmonikusoknak felelnek

majd meg, melynek bemenetei a megfelelő amplitúdó-, és fázisparaméterek lesznek, a kívánt frekvenciát pedig a bázisfüggvények leírásában a következőképpen adjuk meg:

$$f_i = if_0 \; ; \; i = 1,2,....,$$

,ahol az $f_0$az adott síp alapfrekvenciája.

2.2.2.1. Sztochasztikus jelenségek

Jelen esetben az orgona hangját jellemző nem periodikus, sztochasztikus jel elsősorban zajként jelenik meg. Diszkrét idejű generálása szubtraktív módszerekkel lehetséges. Az orgona hangjának jellegzetessége, hogy véges számú, különböző frekvenciájú zajcsúcs összegeként értelmezhető. Ebből következően kézenfekvő megvalósítás lehet az, hogy a zajcsúcsok számával megegyező számú szűrővel kialakítjuk a zajcsúcsokat Gauss-zajból. A zajcsúcsok száma, akárcsak a felharmonikusok száma, hangonként változó értéket vesz fel (A regiszter típusától, magasságától, síp típusától és más tényezőktől függően). Miután a megfelelő zajgenerátorral előállítottuk az additív zajt, hozzáadjuk a periodikus jelhez.

A zajhoz kapcsolódó szűrőparaméterek előállítása az analízis feladata. A szintézishez erről elég annyit tudni, hogy először is a MATLAB-ban írt analízis program a bejövő hangmintánkból előállítja a hang frekvenciatartománybeli képét.[1] Ezen tisztán láthatóak lesznek a spektrumképen egyenlő távolságra elhelyezkedő különböző amplitúdójú harmonikusok, egymás mellett, valamint a legtöbb esetben harmonikusokra illeszkedő zajcsúcsok. A program előbb a harmonikusokat levágja, így kinyerjük a zajspektrumot. A következő lépésben a zajspektrumot átlagoljuk, majd egy csúcskereső algoritmus segítségével meghatározzuk a csúcsok frekvenciáit (f_z) , valamint amplitúdóit (A_z).

Tulajdonképpen a szintézis feladata annyi zajszűrő előállítása lesz, amely Gauss zajból az összes zajcsúcsot előállítja.

A rezonáns zajszűrő átviteli függvénye a következő: [11]

$$W(s) = k\frac{\frac{s}{\omega_0}}{1 + 2\xi(\frac{s}{\omega_0}) + (\frac{s}{\omega_0})^2}$$

$$|\mathrm{W(f)}| = |k|\frac{\frac{f}{f_0}}{\sqrt{(1-(\frac{f}{f_0})^2)^2+(2\xi(\frac{f}{f_0}))^2}};$$

, ahol a k erősítési tényező, a ξ csillapítási tényező és az ω_0 a körfrekvencia. A szűrő fontos tulajdonságai közé tartozik, hogy 6 dB/oktávos meredekséggel éri el az ω_0 környezetét, ott ahol ξ-től függően kiemelkedik a karakterisztika, majd levág -6dB/oktáv értékű meredekséggel. A ξ a szűrő sávszélességét határozza meg.

Tulajdonképpen az analízisnél meghatározott 3 paraméter (amplitúdó (A_z), frekvencia (f_z), és sávszélessége segítségével a másodfokú rezonáns zajszűrő paraméterei már egyértelműen adottak. Adott zajcsúcs sávszélességére (A_{-6}) azt a szélességét értjük, amelyen a zaj értéke nem éri el a zajcsúcs értékének az 50%-át. (A forrás szerint ez az érték 71%, de a pontosság, és jobb zajszűrő tervezése szellemében az analízis ezt az értéket 50%-ra definiálja át.)[1]

Némi számolás után a rezonáns szűrő kívánt paraméterei a következők lesznek:

$$f_0 = f_z$$
$$\xi = \left| \frac{(1 - \frac{f_z}{f_0})(1 + \frac{f_z}{f_0})}{2\sqrt{3}\frac{f_z}{f_0}} \right|$$
$$k = 2\xi A_z$$

Az [xy] egyenletben megadott analóg szűrő együtthatói a fent kapott paraméterekkel a következő:

$$b = [0 \;\; k\omega_0 \;\; 0]$$
$$a = [1 \;\; 2\xi\omega_0 \;\; {\omega_0}^2]$$

Miután ezeket a paraméteket áttranszformáltuk s-tartományból z-tartományba, előáll a modell implementációjában is használatos forma, a másodfokú rezonáns szűrők z-tartománybeli polinom-együtthatóit tartalmazó ad valamint bd vektorok.[10]

2.2.2.2. Tranziens jelenségek

Tranziens jelenségeken általában az időtartományban lévő jel nulla időpontból induló az állandósult állapot eléréséig tartó változását, felfutását értjük, valamint az állandósult állapotból a jel megszűnéséig tartó folyamatot, amit gyakran lecsengésnek is nevezünk. Ezek a változások exponenciális jellegűek, és megvalósításuk (harmonikusonként külön) burkológörbe-generátorokkal történik, amik végtelen impulzusválaszú (IIR) szűrők egységugrásra adott válaszaiként jönnek létre. A modellalkotáskor azért célszerű a szűrők ugrásválaszát és nem az impulzusválaszát felhasználni, mert az orgona állandó gerjesztésű hangszer (ellentétben például a pengetős hangszerekkel), tehát a harmonikusok nem lecsengőek, állandósult állapotba kerülnek.
Mindezek ismeretében az $i.$ harmonikus $k.$ időpontbeli mintájának előállítása:

$$x_{i,k} = h_{i,k} Re\left\{ A_i e^{j(2\pi k \frac{f_i}{f_s} + \varphi_i)} \right\}, k = 0, 1, 2, \ldots,$$

Az A_i, f_i az adott harmonikus amplitúdója és fázisa, a $h_{i,k}$ pedig a harmonikus burkolóját előállító szűrő ugrásválaszának értéke a $k.$ időpillanatban.

A orgonahang harmonikusok és zaj összegeként áll elő. A komponensenkénti elkülönülés tranziens folyamatokra is igaz, ezért célszerű a tranzienseket is komponensenként külön felépíteni. Tehát meg kell határozni a harmonikusok és a zaj tranziens burkolóit. Mivel a sípok berezgése exponenciális jellegű[1], digitálisan IIR- szűrővel modellezhető jól. Ahhoz, hogy a felfutó tranziensek előálljanak a szűrőre egységugrás jelet kell kapcsolni. Ekkor a jel nullából indul, egy idő után exponenciális jelleggel, adott esetben túllövéssel éri el a maximumot, majd esetenként kisebb kilengésekkel eléri az állandósult állapotot, amely körülbelül egységnyi értékű. Az IIR-szűrő z-tartománybeli átviteli függvénye: [11]

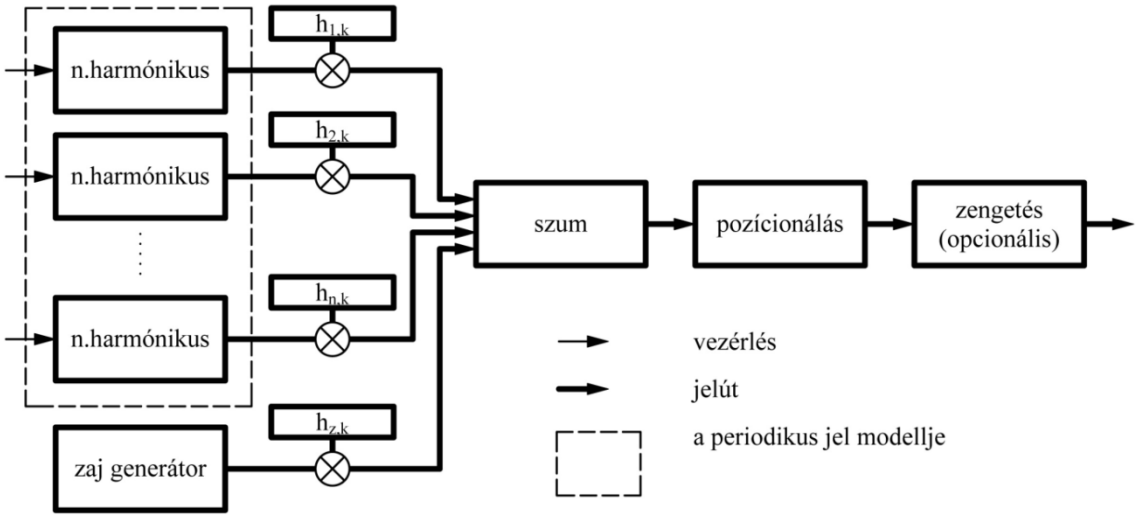
[10] Szintézis modellünkben ezeknek a szűrőknek a száma hangonként 10-18 között változik.

$$W(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_m z^{-m}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_n z^{-n}},$$

ahol n a nevező és szűrő fokszáma, m a számlálóé, b_i, a_i pedig a számlálóban, valamint a nevezőben lévő polinom-együtthatók. Tehát az n meghatározza a szűrő fokszámát, amelynek növekedtével jobban formázhatóvá válik a szűrő, de költségesebbé is. Kísérleti sorozattal megállapítást nyert, hogy harmadfokú IIR szűrővel ideálisan megvalósíthatóak a komponensek burkológörbéi. [1]

2.2.2.3. Az integrált jelmodell

A fenti tényezők figyelembevételével az integrált jelmodell látható a következő ábrán[11] [1]:



8. ábra: Az integrált jelmodell

A szaggatott vonallal határolt területen valósítjuk meg az egyes harmonikusokat, amelyek bemeneti paramétereiként megadjuk az adott harmonikus amplitúdóit és fázisértékeit. Legalul látható az additív zaj előállításáért felelős zajgenerátor, bemenő paraméterei az egyes zajcsúcsokat előállító szűrő paraméterei.

A tranziens burkológenerátorok bemenő paraméterei a megfelelő szűrő paraméterei, a kimeneteik pedig a szűrők egységugrásra adott válaszaik lesznek.

A pozícionálás a külsö tényezők implementálásának a sztereó, vagy többcsatornás hangot előállító része, amely a sípok térbeli elhelyezkedéséből ered. A távolságkülönbségekből eredő térérzet modellezhető többcsatornás jellel, a sípok eltérő hangereje miatt. A falak reflexiói miatt megvalósítandó zengetéshez szükséges paraméterek nem az adott síptól, hanem a csarnok teremakusztikai tulajdonságaitól függnek. Ez utóbbi két tényező megvalósításával a dolgozat a későbbiekben nem foglalkozik.

[11] Az ábra Sum blokkjánál összegezzük a tranziens-burkolóval ellátott harmonikusokat valamint a zajt.

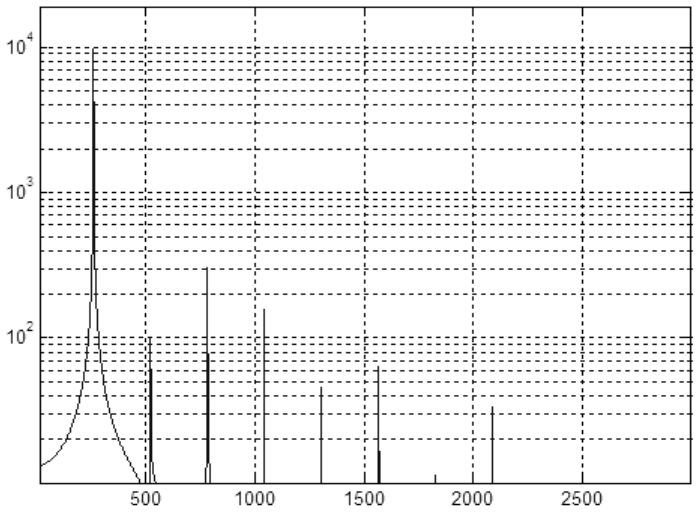
2.3.Orgonahang megvalósítása MATLAB környezetben

A szakdolgozat célja a VST környezetben való Orgonaszintetizátor készítése volt, de a modell algoritmusának tesztelésére kiváló környzetet teremt a MATLAB program, amivel a félév első részében a korábbi orgonahang szintézisről szóló diplomaterv által létrehozott orgonahangok paramétereit közvetlenül fel tudtam használni, és segítségével az integrált jelmodellt létrehoztam.

2.3.1. A paraméterek származtatása

Az F.3.függelékben található mellékletben bemutatásra kerül a szintézishez szükséges paramétereket tartalmazó *.mat* fájl tartalma. Minden fájl egy adott hang paramétereit tartalmazza, a fájl neve az adott hang MIDI-kódja. Egy Principál típusú regiszter egyvonalas C hangja modellezésének a bemutatása következik.
(Egyvonalas C hang: 60.mat. A MATLAB modellezésnél természetesen a fájl betöltése a változók kezeléséhez szükséges.)

2.3.2. Alap- és felharmonikusok generálása



9. ábra: A felharmonikusokkal gazdagított orgonahang spektruma

$$x_{i,k} = h_{i,k} Re\left\{A_i e^{j(2\pi k\frac{f_i}{f_s}+\varphi_i)}\right\}, k = 0, 1, 2, \ldots$$

A fentiekben ismertetett egy harmonikusra vonatkozó diszkrét idejű jelmodell – és az ehhez tartozó képlet – segítségével implementálhatóak az alap- és felharmonikusok. A megvalósítás szinuszos jelek létrehozását kívánja. Az alapharmonikushoz viszonyítva az n. felharmonikus frekvenciája az alapharmonikus frekvenciájának az n-szerese lesz, amit a megvalósításkor

figyelembe kell venni. Az alapharmonikus frekvenciája és az összes harmonikus amplitúdója megtalálható az analízis által létrejött *.mat* fájlból.

2.3.3. Burkolóillesztés tranziens jelenségekhez

A MATLAB implementációhoz szükségünk van a 2.2.2.2. fejezetben ismertetett harmadfokú IIR-szűrők átviteli függvényéből előállított rendszeregyenletéhez:

$$h_i(k) = b_{0,i}e(k) + b_{1,i}e(k-1) + b_{2,i}e(k-2) + b_{3,i}e(k-3) - a_{1,i}h(k-1) - a_{2,i}(3)h(k-2) - a_{3,i}h(k-3)$$

,ahol a $h_i(k)$ az i. harmonikus szűrő válaszának a k. mintája, az $e(k)$ az egységugrás gerjesztés k. mintája, b_i, a_i pedig a fentebb említett i. harmonikus megfelelő polinomegyütthatói. Ezt a műveletet kell elvégeznünk minden harmonikusra és a zajkomponensre a megfelelő paraméterek felhasználásával, majd a $h_i(k)$ vektor előállt mintáit megszorozzuk az i. harmonikus, valamint a zaj megfelelő mintáival, előállítva a tranziensekkel terhelt komponenseinket.

A jelmodell korábbi szimulációknál kiderült, hogy a paraméterek származtatása – és ez alapján történő tranziens burkoló illesztése – a szűrők kimeneteinek csak minden 128. mintájának felhasználását eredményezi.[1] Ez azt jelenti, hogy a végleges tranziens illesztésekor a kapott minták a harmonikus mintáinak csak minden 128. mintájával szorzódnak, közte pedig a lineáris interpoláció megfelelő megoldást nyújt. Az interpolációt és eredményét a következő 2.3.3.1. fejezetben tekinthető meg.

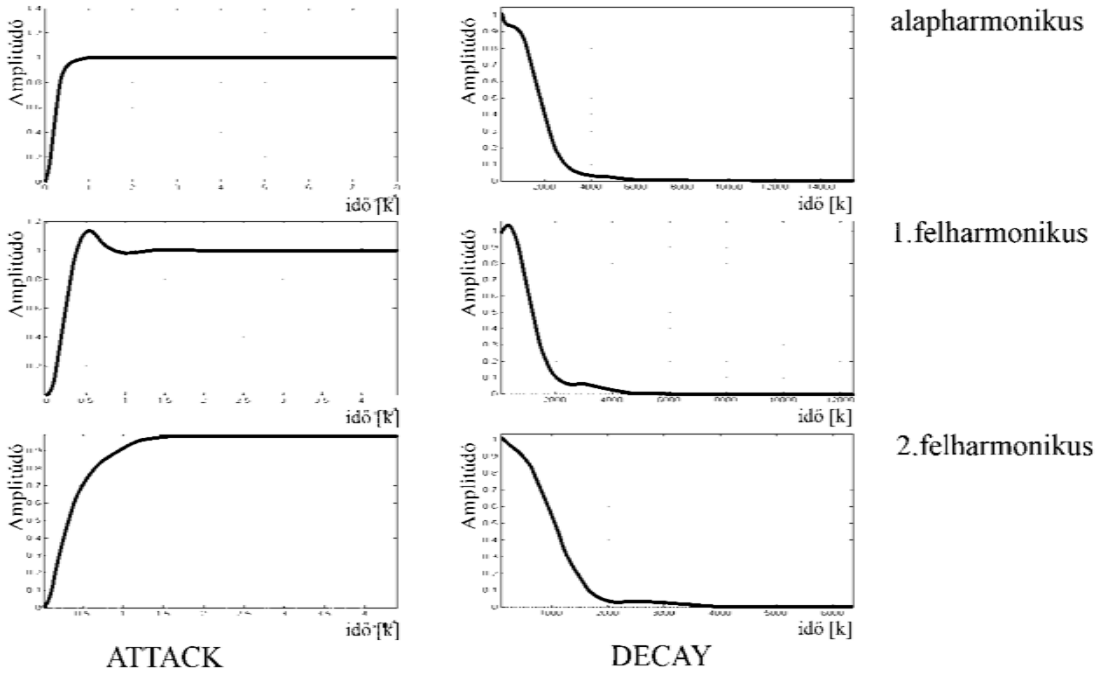
2.3.3.1. Lineáris interpoláció

A tranziens burkológörbéket modellező szűrők a kimeneti értékek azon pontjait határozza meg, amelyeket a harmonikusok jelével szorozva csak minden 128. mintához kapcsolható. Ez a szám az analízis során került meghatározásra. Ezt úgy lehet elképzelni, hogyha nem hajtanánk végre a burkoló két eredetileg kapott szomszédos pontja közötti 128 pontos (adott esetben használt lineáris) interpolációt, akkor az így alkalmazott tranziensek a modell szerint pontosan 128-szor „gyorsabban” lezajlanának, megváltoztatva ezzel a harmonikus amplitúdók állandósult állapotba való beállásának tényleges időállandóit. Gyakorlatban ez a felfutások és lecsengések „kinyújtása”. Az interpoláció célja a szintézisünkben tehát az, hogy időben megfelelő hosszú tranziensek kialakításával kiküszöböljük a hang felépülésének és lecsengésének hiányából adódó „pukkanás” kialakulását.

Ha eddig egy harmonikus állandósult állapotba kerülése mondjuk a [0, T] diszkrét időintervallumban lezajlott (a fenti ábrákon ez néhány száz minta hossznak megfelelő diszkrét időhossz), akkor ez lineáris interpoláció után [0,128T] intervallumra bővül ki, ami már egy-két tízezres nagyságrendnek felel meg, így már valóban érzékelhető lesz a finom felépülése, és lecsengése a lenyomott zenei hangnak.

Az interpoláció a 3.3.4. fejezetben található az orgona szoftver dokumentációjánál, valamint teljes terjedelmű megvalósítása a CD-mellékleten.

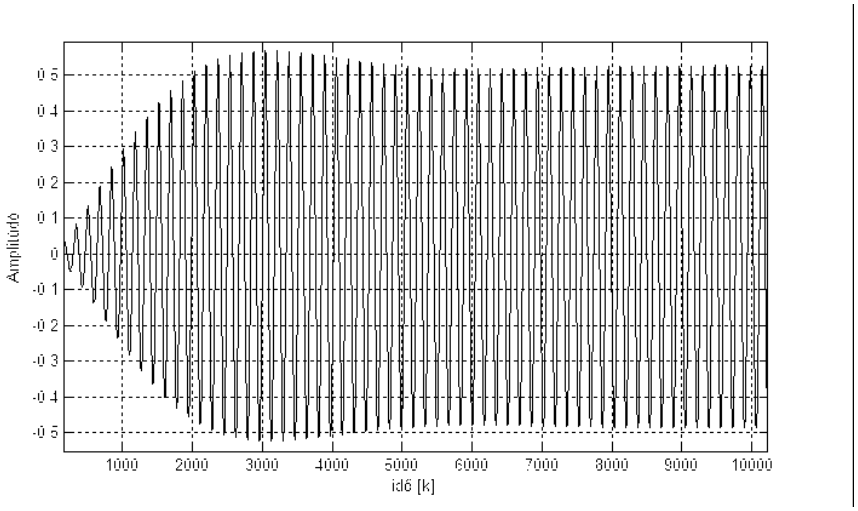
A 10.ábrán látható három harmonikus felfutási és lecsengési tranziense interpoláció után.



10. ábra: A tranziens burkolók felfutása és lecsengése különböző harmonikusokra

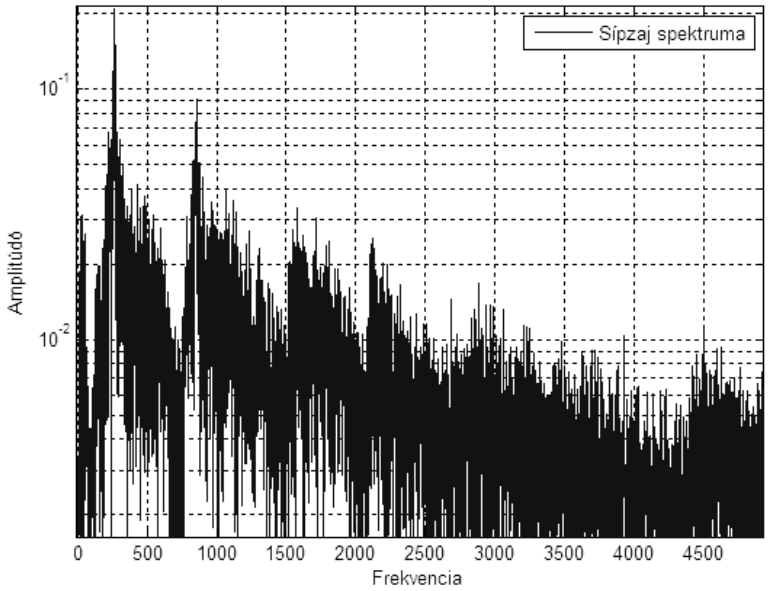
Miután megkaptuk a harmonikusonkénti tranziens-burkolókat, az integrált jelmodell ábrája szerint megszorozzuk az adott harmonikussal, valamint a zajjal, és a komponenseket csak ezután adjuk össze.

Az 10. ábrán a megfelelően interpolált felfutó tranziensekkel ellátott harmonikusoknak az összege látható diszkrét időtartományban. A képen az additív zajt még nem tartalmazó modellben felismerhetőek a tranziensek hatásának kezdeti enyhe túllövése a berezgés folyamatában.



11. ábra: Az alap és felharmonikusok a hozzájuk tartozó felfutó tranziens burkolóval

2.3.4. Zajmodell implementálása



12. ábra: Az egyvonalas C hang előállított zajspektruma

A MATLAB-os integráció – a tranziensek számolásához hasonlóan – megkívánja a zajszűrő, mint IIR-szűrőnknek a rendszeregyenletét leírni, ugyanis az algoritmust implementálása diszkrét idejű.

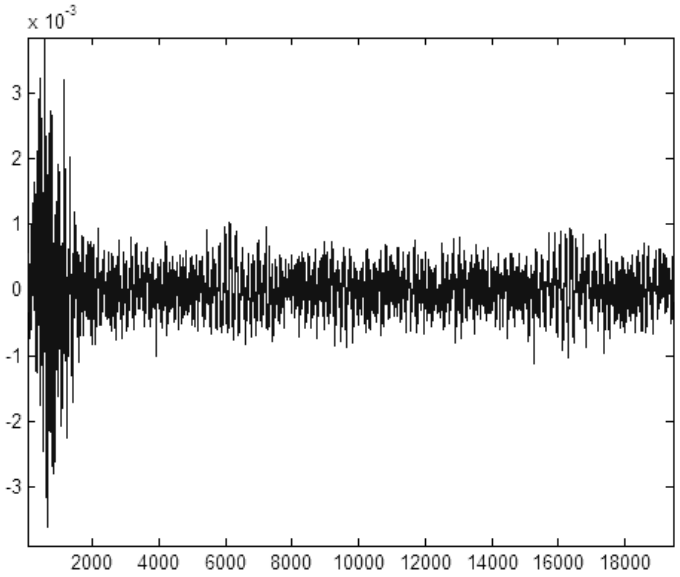
Az i. zajszűrő rendszeregyenlete:

$$p_i(k) = bd_{0,i}r(k) + bd_{1,i}r(k-1) + bd_{2,i}r(k-2) - ad_{1,i}r(k-1) - ad_{2,i}r(k-2)$$

, ahol $p_i(k)$ az i. szűrő sípzajának k.értéke, a k a diszkrét idő, $bd_{0,i}$, $ad_{1,i}$ az i. szűrő diszkrét együtthatókat tartalmazó vektorok különböző elemei, valamint $r(k)$ a bemenetként inicializált Gauss-zaj k.mintája.[12]

Összességében tehát annyi egymás utáni zajszűrőre van szükség, ahány zajcsúcs jelenik meg a sípzaj spektrumában. A példában előállított sípzaj időfüggvénye látható a következő ábrán.

[12] A megvalósítás 18 zajcsúcsot határozott meg. Az így előálló sípzaj spektruma a fenti ábrán, az időfüggvénye tranziens burkolóval pedig a következő ábrán.



13. ábra: Sípzaj időfüggvénye tranziens-burkolóval

2.3.5. Összefoglalás

A szakdolgozatban a külső tényezők modellezésére nem került sor. A MATLAB megvalósítás egy néhány másodpercnyi Principál típusú síp egyvonalas C hangjának a jelmodell alapú hangmintáját generálta.

A harmonikusok megfelelő amplitúdóval ellátva összességében meggyőző volt, véleményem szerint az orgona hangjára már a zaj implementálása nélkül is emlékeztetet. Észrevételként elmondható, hogy a sípzaj nagysága nehezen volt hallható, ezért a szemléltetés kedvéért 20, 40 és 60 dB-es hangerőnövelést követően is tárolva lett *.wav* formátumban a harmonikus komponensekkel, majd önmagában is. A síp hangja zaj nélkül, tranziensekkel, valamint tranziensek nélkül is meghallgatható a CD-mellékleten.

A felfutó tranziens komponensenkéni implementálása után a jelmodell alapú síphang modellje kielégítő eredményt nyújtott.

3. Orgonaszintetizátor plugin megvalósítása

Ebben a fő fejezetben a VST környezetben félév során megvalósított orgonaszintetizátor dokumentálása a cél. A dokumentáció azonban megkívánja, hogy a bizonyos háttérinformációkat is megosszon az olvasóval, amely a fejezet elején kerül bemutatásra. Első részben szó lesz a MIDI protokoll működéséről, majd Virtual Studio Technology bemutatásval folytatódik a témakör, amely kitér az alapfogalmak ismertetésére, némi példát is említve. A 3.2. alfejezet vége a megvalósítás fizikai elrendezését ismerteti, áttérve a félév során fejlesztett orgonamodell dokumentációjára.

3.1. A MIDI protokoll

A MIDI vagy másképpen Musical Instrument Digital Interface egy olyan szabvány kommunikációs protokoll, amely kapcsolatot tud létrehozni szintetizátorok (MIDI billentyűzetek) és más digitális zenei eszközök között. 1980-ban több szintetizátorgyártó által kezdeményezett ajánlás volt, amelyet később szabványszintre emeltek. Elkészítéskor alapvető szempontja volt a gyorsaság, a valós idejű használat miatt.

A MIDI eszközöknek két csoportja létezik, a MIDI-adó és -vevő eszközök. Adatátvitel szempontjából két lényeges tulajdonsága emelhető ki, mégpedig hogy a MIDI interfész olyan aszinkron, soros protokoll, amely 31,25 kbit/s sebességgel képes adatot továbbítani, valamint a másik fő tulajdonság, hogy sorba köthetőek, ami azt jelenti, hogy 16 vevő kiszolgálása lehetséges egy adóval. A MIDI adatok elektronikus tárolására a Standard Midi fájl (SMF) alkalmazható, ami eszközök vezérlésére, tesztelésére nagyon jól használható, de nem ritka a hangszerek rögzített hangjából, arra alkalmas program segítségével MIDI sávokat készíteni, és azokat fájlba menteni.

Egy általános MIDI üzenet két bájtból tevődik össze, ami összesen 16 bit hosszúságot jelent. Az első a státusz bájt, majd ezt követi – általában kettő – adatbájt, amelyeket szabvány szerint értéktartományokra osztottak, majd azokhoz különböző jelentést rendeltek. A MIDI üzenetek általában hexadecimális megjelenítésűek, de értelmezésük binárisan könnyebb, ugyanis a státusz bájt 128-as vagy nagyobb értékkel rendelkezik, egészen 255-ig. Ez binárisan azt jelenti, hogy az első bit mindig 1-es értéket vesz föl. Az adatbitek 0-127 tartományban mozognak, de bővebben a MIDI üzenetek pontos értékeit a következő táblázat tartalmazza.[12]

Éték (decimálisan)	Érték(Hex)	Utasítás	Adat bájtok
128-143	80-8F	Note off	2 (note, velocity)
144-159	90-9F	Note on	2 (note, velocity)
160-175	A0-AF	Key Pressure	2 (note, key pressure)
176-191	B0-BF	Control Change	2(controller no., value)
192-207	C0-CF	Program Change	1 (program no.)
224-239	E0-EF	Pitch Bend	2(least significant, most significant byte)

Tartalma szerint a MIDI adatfolyam első bájtja az úgynevezett utasítás, vagy státusz (*note on*, note off*, pitch bend**), a második a hangmagasság (pitch, vagy MIDI-kód[13]), a harmadik a leütés erőssége (*velocity*). A *note on* egy olyan üzenet a MIDI-vevő számára, ami egy kitüntetett hang leütését definiálja egy adott erősséggel, a *note off* pedig akkor kerül kiküldésre, ha a leütés abbamaradt, vagy leütés történt de a leütés erőssége (*velocity*) 0.

A speciális MIDI utasítások közül a *pitch bend* akkor kerül kiküldésre, ha valamely éppen leütésre kerülő hangnak a magasságát a leütés közben akarjuk változtatni. Ez tipikusan egy külön kis vezérlőt jelent arra alkalmas MIDI billentyűzeten,

A *key pressure* egy leütött billentyűn tartott kéz nyomásával arányos nagyságú információ, ami például zengető effektnél lehet hasznos. A *channel pressure* szintén ugyanezen az elven működik azzal a különbséggel, hogy egy adott csatorna összes leütött billentyűzet lenyomási erősségéből generál jelet, míg a *key pressure* az összes lenyomott billentyűre külön generálódik MIDI információ. Gyakorlatilag ha az alábbi MIDI üzenetek generálására képes billentyűzettel rendelkezünk, nincs akadálya annak, hogy vezérelhessünk többféle paramétert is, mint például zengetést, hangerőt, hangszínt ugyanúgy, mint akár hangmagasságot is.

A *control change* által generált MIDI üzenetek szintén ritkább billentyűzeteknél jellemző. Több csoportja létezik: a csúszkák által vezérelt (amely a paraméterbeállítás széles skáláját nyújtja), a kapcsolók (bináris beállítás, be- és kikapcsolásra alkalmas), és a hangcsatornák külön beállítására szolgáló vezérlők.[13] Ezen csoportba sorolhatók lábkapcsolók, pedálok, *pitch bend* kerekek, modulációs kerekek egyaránt. Egy szemléletes példa *control change* jellegű vezérlésre, mikor az aktuálisan lenyomott hangnak a hangerejét akarjuk szabályozni, vagy éppen kitartani egy hangot, megakadályozva a nem kívánt korai lecsengését.

A *control change* eszközöknek, valamint MIDI adatfolyam protokolljának részletesebb ismertetése túlmutat a dolgozat keretein.

Az orgona szintetizátor működése szempontjából a *note on* és a *note off* jellegű MIDI-üzenetek a legfontosabbak, ezért a továbbiakban ennek az illusztrálása céljából egy egyszerű példa következik:

Az billentyűzet által kiküldött MIDI-üzenet a következő:

9A 60 45

[13] A billentyűzet hangokhoz rendelt MIDI-kódját a Függelék fejezet F.1. ábrája tartalmazza.

, aminek tartalma szerint egy „C” hang (*note on* státusz, 60-as MIDI-kód,) lenyomása következik a 10. MIDI csatornán (A) közepes hangerővel (45-ös *velocity*).

Ha minden hang elnémítása a célünk, megtehető a 7B üzenet elküldésével (*all notes off*), ugyanis előfordulhat fejlesztés során egyéb problémák miatt, hogy a szintetizátorunk hangja „beragad” anélkül, hogy tudnánk az okát. Erre példa, mikor érkezett note on üzenet, de a kábel meghibásodása lehetetlenné teszi a *note off* elküldését, ilyenkor a B0 7B szavakkal némíthatjuk a MIDI vezérelte eszközünket a megfelelő módon, ami mellett második adat bájt kiadása nem szükséges.[12]

3.2. Virtual Studio Technology

A VST egy a Steinberg cég által kifejlesztett, mára már nagy népszerűségnek örvendő szabvány, zenészek, producerek, felhasználók, fejlesztők körében egyaránt elterjedt, akik főként virtuális hangszerek, *effektek* létrehozására, kezelésére használják. A célja egy olyan szabvány létrehozása volt, ami lehetővé teszi a stúdiótechnikai, hangszerelési eszközök könnyű, főleg szoftveres hozzáférését, valamint fejlesztését. Előnye tehát nemcsak az olcsóság, a könnyű hozzáférés, vagy a széles paletta, hanem az egyszerűbb, könnyen beletanulható fejlesztésben rejlik, valamint nem elhanyagolandó szempont, hogy egy ilyen terméket manapság már bárki létrehozhat, ha jól tud programozni és nem kell hardvert építeni hozzá.

A VST egy olyan szoftver interfész, ami alapvetően digitális jelfeldolgozási ismeretek alapján beépített szoftver audio szintetizátor, és különböző effekt *plugin*okkal van ellátva, amelynek a kezelését egy a *host* oldalon lévő szoftver végzi. A központi szerepet a bővíthetőség nyújtja, ami végtelenül sok kompatibilis plugint jelent szerte a világhálón, de a Steinberg önmagában is rengeteg elérhető funkciót, és lehetőséget kínál. A következőkben lesz szó a VST pluginról általában, majd általános bemutatásra kerülnek a VST hangszerek, a VST effektek, a VST gazdaprogramok és legvégül a VST fejlesztői interfészével a Software Development Kit (SDK) -tel ismerkedhetünk meg.

3.2.1. A VST plugin

A VST környezetben megírt kód lefordításával tehát egy *.dll* fájl kiterjesztésű plugin generálódik. Ez bemásolandó a host program pluginokat tartalmazó mappájába, ezután indítható el a host program, amely az elindítással inicializálja a plugint. A VST pluginok egy digitális audio munkaállomáson belül futnak, miközben ellátják a host alkalmazást különböző funkciókkal. A pluginok többsége VST hangszer (VSTi) vagy VST audioeffektként létezik, de léteznek egyéb kategóriák is. Általában a pluginokat saját grafikus felülettel látják el (GUI), amely része az SDK-nak, tehát külön forrásfájlokban találhatók és továbbfejleszthetők. Ezek a host programokban megjelenő felületek, amelyek különböző paraméterek állítására valók, különböző gombokkal, csúszkákkal, potméterekkel, de van néhány, főleg régebbi plugin, amelyek magát a host alkalmazás beépített grafikus felületét használják.

3.2.2. A VSTi

A VSTi-k magukban foglalhatnak jól ismert hardver vagy minta szoftver emulációkat, amelyek jól utánozzák a kinézetüket és a karakteres hangjukat. A VSTi-k alapvető tulajdonsága a MIDI vezérlés. MIDI üzeneteket vár a plugin, ennek segítségével „generálja” a programkódjában definiált hangot attól függően, hogy a MIDI üzenet milyen karaktereket

tartalmaz. Ez a 3.1. fejezetben bemutatásra került.

A VST hangszer tehát audio adatmintákat generál. Jellemző tulajdonsága, hogy a hostba betöltött hangszereken csúszkákkal, potméterekkel változtathatunk szűrő-, hangerő-, frekvencia-, jelalak paramétereket, valamint effektekkel színesíthetjük a hangunkat, de a komolyabb szintetizátorokon még sok más beállítás is elérhető. Az elsők között megalkotott komolyabb VST hangszer a Neon VSTi volt, amely a Steinberg Cubase gazdaprogramban volt használható, ahogyan a Native Instruments' Pro-53-as virtuális szintetizátor is, amely lemodellezte több, a múltban nagy sikerű szintetizátor hangját, és kinézetét egyaránt [16].

3.2.3. VST effektek

A VST effektek, mint például *Overdrive, Reverb* vagy *Phaser* a bejövő audio mintákat dolgozzák fel. Egy olyan dobozként tudjuk elképzelni őket, amelynek meghatározott audio bemenete van, és meghatározott kimenettel rendelkeznek. Általában egy, vagy két kimenetű effekteket fejlesztenek a kétcsatornás hangzás népszerűsége miatt. Léteznek megfigyelő effektek is, amelyek csak vizuális visszajelzéssel szolgálnak, anélkül, hogy „belenyúlnának" az audio adatfolyamba. Legtöbb VST alkalmazás támogatja az „adatláncolást", amelynek segítségével az egyik alkalmazás adatot ad át a másiknak, és fordítva, tehát több alkalmazás sorba csatlakozását. Pl: Egy szintetizátor audio mintái egy *Reverb* effekt bemenetei lesznek.

A VST.SDK alapértelmezetten tartalmaz Gain és Delay effektet, amik egyszerűségük miatt nagy segítséget nyújtanak a VST programfejlesztés alapjainak megértésében.

3.2.4. A VST host

[14] A VST gazdaprogramok[14] olyan szoftveralkalmazások, amelyek lehetővé teszik a VST pluginok betöltését és vezérlését. A gazdaprogram felelős a digitális audio adatfolyam kezeléséért, irányításáért a VST plugintól a hostig mindkét irányban. A host tehát elképzelhető egy kapcsolatként a plugin és a számítógép között. Ezt a kapcsolatot szoftveresen adott Steinberg által megírt forrásfájlok függvényei végzik, amelyeket röviden csak interfész fájloknak hívunk. Ezek megtalálhatók a VST Software Development Kit-ben is (VST.SDK), amit a 3.2.5-ös következő fejezet részletesebben is bemutat.

Minden felhasználói paraméter (közvetlen vagy közvetett), amik a host által módosításokat eszközölnek az audio adatmintákon 32 bites lebegőpontos változók. Törvényszerűen értékeik a 0.0-1.0 tartományba esnek. A VST pluginok által feldolgozott adatminta értékek szintén 32 bites változókban kerülnek eltárolásra, melyek értékei a -1.0 és 1.0 között változhatnak.[15]

A PC-k teljesítményének növekedése törvényszerűen a valós idejű audio pluginok használatát eredményezte. Ehhez szükség van ASIO (Audio Stream Input/Output) protokollt támogató hangkártyákra. Ez egy többcsatornás audio átviteli protokoll, amelyet a Steinberg fejlesztett ki, de mára már több gyártócég átvett és implementált a termékeiben.

A pluginok többsége ingyenes, de például egy igazán jó minőségű szintetizátor pluginért 15-35-50 amerikai dollárt is elkérnek különböző internetes portálokon. Igazán jó VST szintetizátorokat már használtak filmek elkészítésekor, stúdiókban ugyanúgy, mint profi

[14] Néhány gazdaprogram: Cantabile (ingyenes VST gazdaprogram, ami a modulok önálló futtatására használható), MadTracker, Renoise, Fruity Loops, Apple GarageBand, Sonar, Cubase, Logic, Nuendo

zenészek kocerteken vagy híres dj-k élő fellépésekkor.

Összefoglalva elmondható, hogy a VST piacon megjelenése új lehetőséget nyitott azok számára, akiknek eddig a magas színvonalú virtuális hangszerek áraik, vagy méreteik miatt elérhetetlenek voltak. Napjainkra azonban az otthoni virtuális stúdiók létrehozása mellett egy nagy szoftverfejlesztői felületet teremtett a zeneipar iránt érdeklődök számára.

3.2.5. A VST fejlesztői felület (VST Software Development Kit)

A VST.SDK egy olyan nyílt forráskódú fejlesztői fájlok összessége, amelyet a Steinberg lehetővé tette a harmadik fél általi ingyenes plugin fejlesztést, valamint azok gazdaprogramokkal való használatát. A fejlesztői környezet alapvetően objektum-orientált C++ programozási nyelven írt osztályokat és függvényeket tartalmaz. A VST.SDK elérhető mindhárom (Windows, Linux, Apple) platformon, de a legtöbb alkalmazás csak Windows és Apple számítógépeken.

Az F.2. mellékletben megtalálható a VST.SDK 2.4 könyvtárszerkezete. A következőkben rövid bemutatásra kerül a egyszerű VST hangszer, az SDK beépített példa-szintetizátor fájlszerkezete, majd a fejlesztői környezet interfésze a félév során megvalósított orgona szintetizátoron keresztül.

Az SDK szintetizátor mintaplugin forrásfájljai a **vstsdk2.4\public.sdk\samples\vst2.x\vstxsynth\win** linken érhetők el, ahol a *vstxsynth.proj* fájlt szükséges megnyitni. Megnyitás után minden alkalommal két fő mappában helyezkedik el a fejléc, valamint a fejlesztői fájlok. Minden plugin alapértelmezetten kell, hogy tartalmazzon interfész forráskódokat, amelyeket az **Interfaces**-ben található *aeffect.h, aeffectx.h*, és *vstfxstore.h* fájlok tartalmaznak. Ezek a Steinberg által előre megírt fő osztályokat, és függvényeket tartalmazzák, amik a host és plugin közti kommunikációt, adatvezérlést, inicializálásokat, engedélyezéseket, és tiltásokat bonyolítják le. A VST plugin fejlesztőinek ezeket semmilyen körülmények között nem szabad módosítaniuk, valamint a fejlesztéshez ezen függvények ismerete nem szükséges.

A Source Files mappa két fő részből áll. Az egyik része a **vst 2.x** mappa, amely a 2.0-ás fejlesztői csomag interfész fájlokat tartalmazza. Új verzió kiadásánál ezek módosításra kerültek, vagyis az x-szel végződődő fájlok olyan osztályokat tartalmaznak, amelyek leszármazottjai az `AudioEffect` ősosztálynak. Ez a jelenség a VST.SDK 2.0 verzió óta áll fenn, ebből következően a fejlesztett plugin osztálya (adott esetben `VstXSynth`) az `AudioEffectX` osztályból van származtatva. *A mappa tartalma: aeffeditor.h, audioeffect.cpp, audioeffect.h, audioeffectx.cpp, audioeffectx.h, vstplugmain.cpp.*
A fejlesztői felület a **vst2.x** mappa *vstxsynth.cpp, vstxsynth.h, vstxsynth.proc.cpp* fájlokból áll.

3.2.6. Az elrendezés

A működés elrendezése a következő. Szükségünk van egy MIDI billentyűzetre, egy megfelelően csekély késleltetésű hangkártyára, ami a PC-ben van, és csatlakoztatva van a billentyűzethez. A számítógépen lévő VST host, adott esetben Cubase-ben betöltött aktív VST szintetizátor pluginnal rendelkezünk. Az algoritmus megértéséhez feltétlenül szükséges, hogy tisztában legyünk azzal, mi történik egy VST hangszer megszólalásakor, tehát valós idejű működéskor.

Röviden összefoglalva a következő történik: Ha egy billentyű lenyomása megtörténik, MIDI információ érkezik a kis késleltetésű hangkártyán keresztül, amelynek feldolgozása a megírt függvények alapján a host feladata. A beépített hangkártyák mintavételi frekvenciája

44100 Hz, ami 44100 feldolgozott mintát jelent másodpercenként. A MIDI kód tartalmazza a leütés erősségét (velocity), valamint a hang magasságát (pitch). Ezek alapján a program felépíti a jelet, tehát a megfelelő hangmagasságból kikeres egy ehhez tartozó frekvenciaértéket egy előre feltöltött táblából, és adott frekvencián megszólaltatja az inicializált jelalakot, amelynek meghatározott hossza van (a példában 4096 minta, ami kis közelítéssel a másodperc tizedrészével egyenértékű a mintavételi frekvencia miatt). Ez a meghatározott hossz a másodperc töredéke alatt lejátszódik, de az állandósult állapotban a plugin feladata, hogy ezt a hurkot folyamatosan ismételgetve játssza le (loopolja)[15]. másik lényeges MIDI-információ a velocity, amely egy paraméterértéket fog tartalmazni 0-1 tartományban (a 3.2.4-es fejezetben leírtak alapján) a megfelelő maszkolás után. (Ez a hangerőért felel, ami orgona szempontjából nem lényeges, ugyanis a billentés erőssége orgona esetében érdektelen) . [15]

A program által generált additív zaj hozzáadódik a kimeneti jelünkhöz, ami nem szakaszonkénti ismétléssel hajtódik végre (sztochasztikus jelenség révén), hanem állandóan új minták keletkeznek.

A tranziens burkolók jelfolyamhoz kapcsolódása komponensenkénti egyszerű szorzással történnek, mivel a jel jelentős amplitúdóváltozásait a lenyomott hang elején és végén hajtja végre, a köztes állandósult állapotban nem változnak.

A továbbiakban a fenti rövid összefoglalás részletes kifejtésére kerül sor, a függvények működésén keresztül.

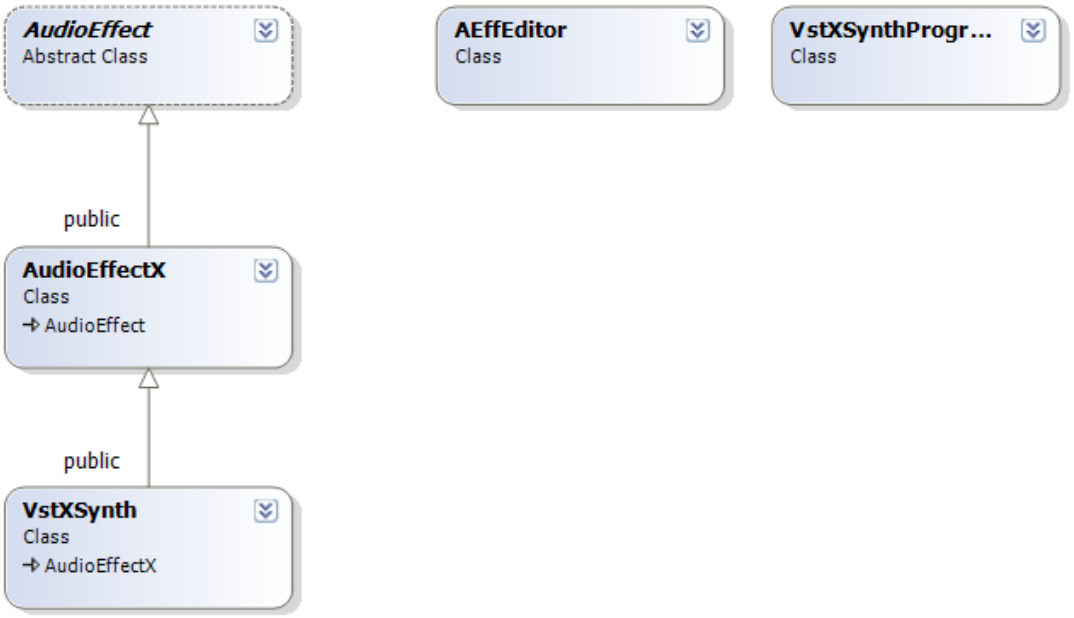
3.3. Orgonahang szintézise VST környezetben

Az előzőekben bemutatásra láthattuk, hogyan épül fel általános mintaplugin, és milyen fejlesztői és interfész fájlokat tartalmaz. A továbbiakban az orgona hangját előállító szintetitizátor plugin működésének dokumentálása a cél. Ez a szintetizátor példa nagyon jó lehetőség arra is, hogy megértsük, miképpen épül fel egy közepes nehézségű VST plugin, ugyanis a program függvényszerkezetét tekintve nem tér el az SDK szintetizátor példaprogramjától, ebből kifolyólag megláthatjuk milyen lehetőség van „belenyúlni” egy VST szintetizátor algoritmusába. Terjedelmi okok miatt a fejezet nem dokumentálja a fejlesztett kód minden részletét, de a félév során történő fejlesztésnek minden fontosabb része megmagyarázásra kerül. A teljes, kommentekkel kiegészített forrásfájlok a CD-mellékleten megtalálhatóak, az ezekből generált *.dll* fájllal együtt.

A hangszer önmagában két kimenettel rendelkezik és polifonikus, tehát egyszerre (beállítástól függően) több billentyű lenyomását támogatja. A programkódban függvények szerint elkülönülnek a harmonikusok generálásáért és kezdeti inicializálásokért `(initProcess ())`, a kimeneti jel előállításáért (`processReplacing()`), valamint a MIDI események kezeléséért (`processEvents()` felelős legfőbb részek, a paraméterbeállító, grafikus felülettel, és annak különböző paramétereivel foglalkozó résztől. Az előbbi a *vstxorgonaproc.cpp*, az utóbbi a *vstxorgona.cpp* fájlban lett megvalósítva. Utóbbi fájtl függvényei részletesen nem kerülnek bemutatásra. A program bemutatása a konkrét algoritmust is tartalmazó *vstxorgonaproc.cpp* dokumentációjával kezdődik.

[15] A pontos leírás kedvéért: A hurkok tartalmazzák az alap- és felharmonikusokat egyaránt, melyeknek paraméterei és száma a modellezendő hang (leütött billentyű) függvényében változó.

A következő ábrán az orgona plugin osztálydiagramja látható[16].



14. ábra: VST plugin osztálydiagram

Az SDK (3.2.5) fejezetében megemlítésre került az `AudioEffectX` és `AudioEffect` osztály kapcsolata, amelynek egymásból való származtatása a 14. ábrán megfigyelhető. A fejlesztői környezetet alkotó fájlok két (egymással „friend" kapcsolatban álló) osztály, a `VstXSynthProgram` és a `VstXSynth` tartalmazza, amelyek közül az előbbi konstruktora a GUI paramétereit tartalmazó változók kezdeti értékadását végzi. Ezeknek a későbbi értékadásáért a `VstXSynth`–ben felsorolt függvények a felelősek. Ez úgy értendő, hogy a hostban megjelenő program grafikus interfészében (GUI) lévő paraméterváltozók alapbeállítását a `VstXSynthProgram` osztály végzi (például a plugin alapértelmezett hangerő beállítása). Működés szempontjából a főosztálynak a `VstXSynth` osztályt tekinthetjük, mert a függvények nagy része (beleértve a működést végző függvényeket is) itt lettek felsorolva. A 3.2.5. fejezetben is megemlített `AudioEffectX` leszármazottjaként kezeljük a `VstXSynth` osztályt, amely az `AudioEffect` ősosztály „unokájának" is mondható.

A következőkben a `VstXSynth` által tartalmazott legfőbb függvények működésének részletei következnek.

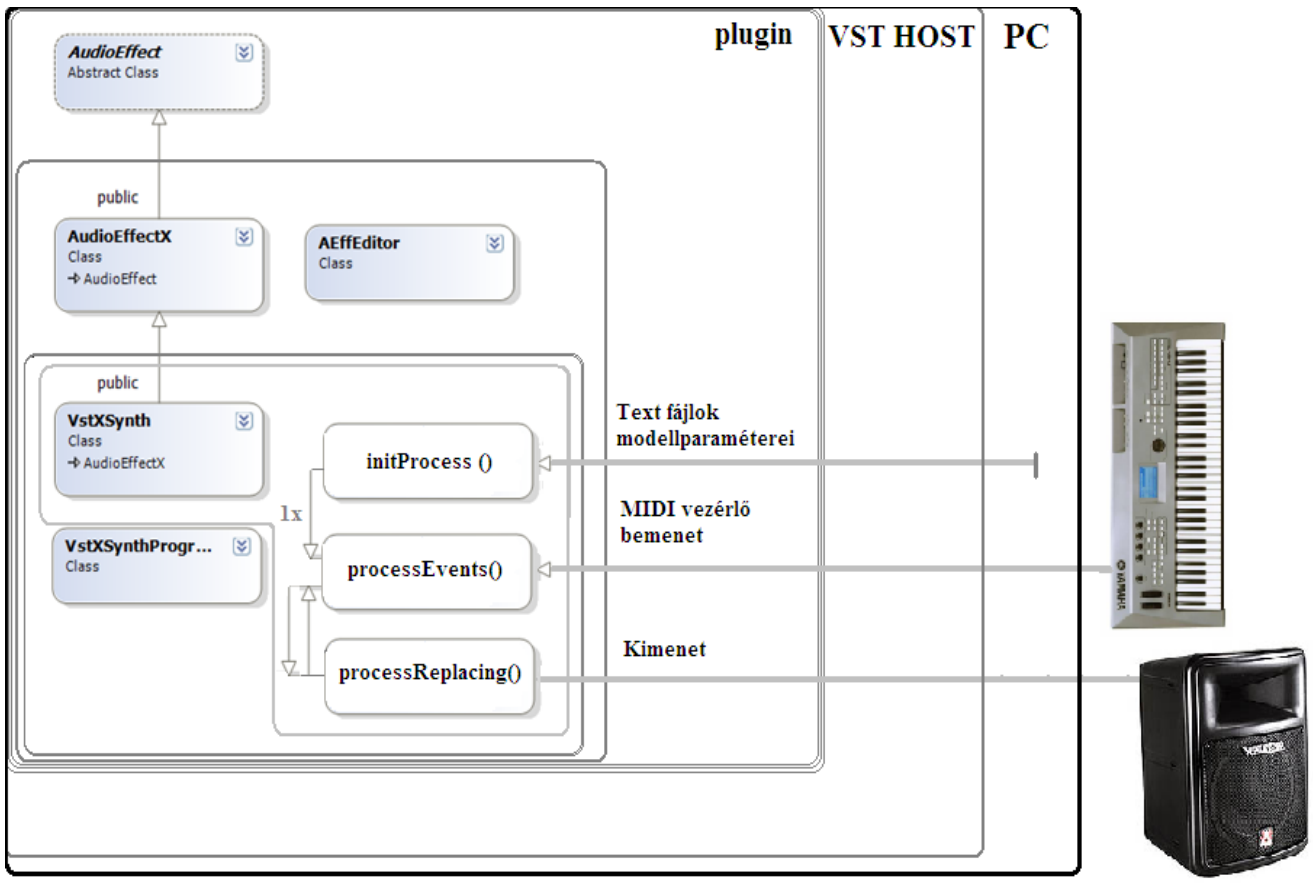
Minden VST szintetizátor tartalmaz egy `processEvents()` és egy `processReplacing()` függvényt (bizonyos pluginok `process()` –t is tartalmaznak[17]), amik közül az előbbinek a MIDI adatfolyam kezelése a feladata, utóbbinak pedig a vezérléssel szinkronban a kimeneti értékek generálása. Megfogalmazható úgy is, hogy a két függvény

[16] A részletes típusdefiníciókat, struktúrákat, valamint azok kapcsolatát bemutató osztálydiagramok a mellékletben találhatóak meg.

[17] A processReplacing() és process() függvény a régebbi verziókban (pl SDK.2.3) egyszerre volt megtalálható. A kettő közötti különbség annyi, hogy míg a process() függvény a kimeneti értékeket hozzáadja a már meglévő kimeneti puffer értékeihez, a processReplacing() pedig felülírja azokat.

szoros kapcsolatban áll egymással (mindig egymás után hívódnak meg). Ez úgy képzelhető el, hogy a meghívott `processEvents()` a pillanatnyi MIDI eseményeket tárolja, majd a közvetlenül utána meghívott `processReplacing()` az előzőek alapján tárolt „bemenetek” alapján generálja a kimeneteit. Az ily módon definiált kimenetek generálása blokkonként történik, ahogy a bemenetek által tárolt MIDI események is blokkonként érkeznek, többek között azzal az információval (deltaFrames) is, hogy a blokk mely pozíciójában történt a változás. Egy egyszerű példa: Pontosan a blokk közepén történt a billentyű lenyomása (egyvonalas C), amelyet a `processEvents()` észlel, és eltárolja az információkat a hangról(note=60, velocity=adott billentési sebesség, deltaFrames= a blokk hosszának a fele). Ezután meghívódik a `processReplacing()` függvény ami nullákat ad ki a blokk első felében (deltraFrames pozíció előtti blokkrészben), a második felében pedig az orgona egyvonalas C hangjánál meghívandó – és ehhez tartozó komponensek generálásához szükséges fájlból olvasott paramétereinek a feltöltésével – legenerálja a síp hangját, ami annyi blokkon (ebből következően egymásután hívott függvények lefutásán) keresztül tart, amíg nem érkezik meg a MIDI „note off” információja, ami az elnémításra kiadott parancs. A note off típusú utasítást a hanghoz kapcsolódó változók kinullázása követi (note=0, velocity=0). A fenti gondolatmenet megértése a legfontosabb a dokumentáció megértése szempontjából.

A `processEvents()` tehát a MIDI adatfolyam kezelését végzi. aminek a átláthatóságához szükséges a 3.1. fejezetben lévő MIDI adatfolyam ,valamint a következő fejezetben lévő VST Events módszer ismertetése. A fent ismertetett folyamat illusztrálását a 15. ábrán látható digitális modell végzi.



15. ábra: A valós idejű plugin működése

3.3.1. A VST Events

A VST Events olyan mechanizmus amely kifejezetten különböző tipusú vezérlések kezelését teszi lehetővé, folyamatos kapcsolatban állva a `processReplacing()` függvények adatfeldolgozást végző pufferével. A következőkben bemutatásra kerül a VST Events módszernek pár adatstruktúrája: [15]

```
struct VstEvent
{
    long  type;
    long  byteSize;
    long deltaFrames;
    long  flags;
    char  data[16];
};
```

A `VstEvent` struktúra egy bekövetkező általános vezérlő eseménynek az adatszerkezete figyelhető meg. a `type` változó az események különböző fajtáit jelenti, mely az enum adatszerkezetben láthatóan MIDI, Audio, Video, Paraméter, Trigger, valamint egyéb eseményként értelmezhető. A `byteSize` az esemény elfoglalt mérete, a `deltaFrames` az adott blokkban elhelyezkedő esemény kezdeti pozíciójára utal (már említett blokkon belüli pozíció, ahol az esemény aktivizálódott). A `flags` nem definiált általános változó, valamint tartalmaz a struktúra egy általában 16 karakteres adatvektort, az esemény vezérlőinformációinak a tárolására (`data[16]`).

```
enum
{
    kVstMidiType  = 1,
    kVstAudioType,
    kVstVideoType
    kVstParameterType,
    kVstTriggerType
    //  ...etc
};
```

Ha az adott eseménytípus értéke 1, akkor tekinthető aktívnak. A szintetizátor mintaplugin szempontjából a `kVstMidiType` flag aktív értéke az érdekes, ami a `processEvents()` fügvvényben kerül vizsgálatra, a többi esemény bemutatása túlmutat ezen dolgozat keretein.

A VST Events módszer bevezet egy olyan struktúrát is, amely az aktuális audio mintákat tartalmazó blokkokban lévő eseményeket írja le, vagyis több esemény tárolását végzi, ugyanis a VstEvent struktúra példányait is tartalmazza, kiegészítve más változókkal. Ez a `VstEvents` struktúra:[15]

```
struct VstEvents
{
    long  numEvents;
    long  reserved;
    VstEvent* events[2];
};
```

Létezik egy csak MIDI esemény tárolását végző `VstMidiEvent` struktúra is, ami tartalmában

hasonlóságot mutat a fent tárgyalt `VstEvent` struktúrával, kiegészülve a note-ot leíró változókkal. A esemény típusváltozója (`long type`) a korábbiakkal összhangban `kVstMidiType` típusú lesz, ami MIDI eseményt jelent. Mérete 24 bájt (byteSize). Ezen struktúra példányai lesznek a MIDI események tárolói, amelynek adatai a `char midiData[4]` vektorban kerülnek eltárolásra (a vektorban 1-3 helyiérték van lefoglalva erre a célra).

```
struct VstMidiEvent
{
    long  type;
    long  byteSize;
    long  deltaFrames;
    long  flags;
      .
      .
    char  midiData[4];
      .
};
```

3.3.2. A processEvents függvény

A Vst Events módszer, és a MIDI-kódok jelentésének ismeretében bemutatásra kerülhet, hogyan működhet az orgona szintetizátor MIDI kezelése polifónikus esetben.
A MIDI információ feldolgozását a fent említettek alapján tehát a `processEvents()` végzi, mely a következőképpen néz ki.

```
VstInt32 VstXSynth::processEvents (VstEvents* ev)
{
for (VstInt32 i = 0; i < ev->numEvents; i++)
      {
      if ((ev->events[i])->type != kVstMidiType)
            continue;

      VstMidiEvent* event = (VstMidiEvent*)ev->events[i];
      char* midiData = event->midiData;
      .
      .
      .
```

A VST Evenst módszer alapján átlátható módon az `ev->events[i]`-ben tárolódnak az események, és miután ez átment a típus szűrésén (tehát MIDI esemény érkezett), folytatódik a program. A for ciklus addig fut le, amennyi beérkezett MIDI esemény van, felhasználva ezeket a `processEvents()` meghívásakor. Érdekesség, hogy mivel a függvény másodpercenként többször hívódik meg, akkor lesz egy meghívás alatt több esemény, ha majdnem egyszerre, vagy időben egymáshoz nagyon közel vannak a billentett hangok bejövő információi. A `continue` után egy pointert definálunk, amely a MIDI adatot tároló vektorra (`midiData`) mutat. Ennek a 0. eleme a `status` változóba, 1. eleme a note (MIDI-kód), 2. eleme a velocity (leütés sebessége) nevű változókba kerülnek a megfelelő hexadecimális számmal maszkolva. A maszkolás azért szükséges, hogy az egyértelmű azonosításhoz fontos biteket megtartjuk, a többi elhagyásra kerül, mint ahogyan a `status` bit is csak így válik vizsgálhatóvá. Ez az alábbiakban hajtódik végre:

```
      VstInt32 status = midiData[0] & 0xf0;
```

```
if (status == 0x90 || status == 0x80)
{
	VstInt32 note = midiData[1] & 0x7f;
	VstInt32 velocity = midiData[2] & 0x7f;
```

Az `if` ágba történő belépés feltétele, hogy *note on* vagy *note off* „üzenet” érkezzen, mert a következőkben csak ezek fontosak a szintetizátor plugin szempontjából, és ezeknek a kezelése is van részletesen kifejtve (bonyolultabb szintetizátor funkcióira nem optimalizált). A következőkben külön feltételvizsgáló elágazások értékelik ki a *note on (0x90)*, valamint a *note of (0x80)* státuszú adatokat.

```
if (status == 0x90)
	{
		if (velocity != 0)
		{
			noteOn (note, velocity, event->deltaFrames);
		}
		else if (velocity == 0)
		{
			noteOff (note, event->deltaFrames);
		}
	}
	if (status == 0x80) noteOff (note, event->deltaFrames);
```

Ha ugyanis a státusz bájt *note on*, és a `velocity` értéke nem nulla, meghívódik a `noteOn()` függvény, vagyis beszélhetünk megszólaló hangról, ha viszont a billentés sebessége 0 értéket kap, nem lesz hallható hangunk, ahogyan a *note off* státusz érkezésénél sem. Ez széles körben használt módszer, ami alapján kétféleképpen némulhat el egy hang.
A `processEvents` függvény kezeli a *control change* státuszt is, aminek detektálását követően az összes hang elnémul.

A következőkben a `noteOn()`, `noteOff()` bemutatása szükséges, amely a polifónikus billentési képességet kezeli. Amikor hangokról beszélünk, egy struktúra példányáról beszélünk, melynek változói a hang sajátosságait hordozza, tehát feltöltődnek minden aktivizálódásukkor, és lenullázódnak minden „elengedéskor”.
A `hangok` típusú struktúra a következőképpen néz ki[18]:

```
struct hangok {
	//A zajgeneráláshoz megfelelő g változó és vektorok
	float random[Size];
	float noise[Size];
	float feedbuff[Size];
	float sum[Size];
	int g;

	//hangra jellemző értékek!!!
	VstInt32 note;
	VstInt32 velocity;
	VstInt32 deltabe;
	//a jel pillanatnyi fázisát tárolja
	float fPhase1;
	//ha szól a hang true, ha nem false
```

[18] Az indexelt változók a harmonikusokra külön tárolják az adatokat. Terjedelmi okok miatt nem lett feltüntetve az összes.

```
            bool isOn;

            //A tranziens-burkolókhoz és interpolációhoz megfelelő változók
            float y_old1;float y_old2;…….;float y_old4; float y_old26
            //pillanatnyi interpolált burkoló minta értékét tároló változó
            //az n. harmonikusra
            float yip_out1….. float yip_out26 ;
            //pillanatnyi interpolált zaj burkoló minta értékét tároló
            változó
             float zajip_out;
            //tranziens segédváltozó
            int flag_128;
            //egységugrás bemeneti változók
            int h0;
            int h1;
            int h2;
            int h3;
            //tranziens rendszer visszacsatolt bemeneti értékeit tárolják

            float y_1;
            float y_2;     float y_3;…….float y_26;
            float y1_1;     float y2_1;     float y3_1;
                   .
                   .
            float y1_26;     float y2_26;     float y3_26;
                        }
hangok hang[6];
int MAX_POLIFONIA = 6;
```

A hang egy hangok struktúra típusú példány, amely a beállítás alapján 6 párhuzamos csatorna megszólalását teszi lehetővé. Ha egy billentyű lenyomásra kerül, akkor egy példány változói a struktúra szerinti változóknak ad értéket, egyes változói a MIDI kezelés függvényekben (velocity, note, deltabe) töltődnek fel, a többi pedig a `processReplacing()`-ben[19]. A MIDI-kezelés szempontjából most a `note`, `velocity` és `deltabe` változók a fontosak (amelyeket a függvény paraméterként kap), ugyanis a `noteOn()`, `noteOff()` ismertetése következik:

```
void VstXSynth::noteOn (VstInt32 note, VstInt32 velocity, VstInt32 delta)
{
      n_i=0;
      for(int l_i=0;l_i<7;l_i++)
      {
            for(int M_in=MAX_POLIFONIA;M_in>1;M_in--)
            {
            if((hang[M_in-1].isOn==true) && ((hang[M_in-2].isOn == false)
|| (hang[M_in-2].velocity == 0)))
                  {
                        hang[M_in-2].note=hang[M_in-1].note;
                        hang[M_in-2].velocity=hang[M_in-1].velocity;
                        hang[M_in-2].deltabe=hang[M_in-1].deltabe;
                        hang[M_in-2].isOn=true;[20]
                        .
                        .
```

[19] Utóbbi kijelentés pontosításra szorul, ugyanis az inicializáláskor a `hang[]` tömb minden elemének változói kezdeti értéket nyernek, de a `processReplacing()`-ben az értékek folyamatosan felülíródnak.

[20] Minden változó nem lett terjedelmi okokból feltüntetve.

```
                        hang[M_in-1].isOn=false;
                        hang[M_in-1].velocity=0;
                        .
                        .
                }
        }
}
```

A `noteOn` függvényt három fő részre lehet osztani. Az első részben a lenyomott hangokat tartalmazó vektor elemeinek a rendezése a cél. Ez azért szükséges, mert tegyük fel, hogy egyszerre tartunk lenyomva 6 különböző hangot, és elengedjük a legutoljára, és a másodjára leütöttet, akkor „hézagok” keletkeznek a `hang[]` vektor-ban, ami a később bejövő hangok vektoron belüli elhelyezése szempontjából elég bonyolult lenne. Így első körben minden függvénymeghíváskor rendezzük a vektor, így az első n letartott hang, az első n helyet fogja feltölteni, így könnyen követhető válik az algoritmus.

```
for(int e_i=0;e_i<MAX_POLIFONIA;e_i++)
{
        if(hang[e_i].isOn==true)
        {
                if(e_i==MAX_POLIFONIA-1)
                {n_i=0;}
                else
                {n_i++;}
        }
}
```

A második szakasz az `n_i` index megfelelő beállítását végzi, amely a beérkező következő hang helyét azonosítja a `hang[]` tömbben,

```
hang[n_i].note=note;
hang[n_i].velocity=velocity;
hang[n_i].deltabe=delta;
hang[n_i].isOn=true;
}
```

ami a harmadik szakaszban lévő algoritmust helyes működésének feltétele, így mindössze a rendezett, és jól beállított indexű helyre másolja a bejövő `note`, `velocity`, `delta` értékeket, valamint az `isOn` flaget `true` értékűre állítja.

A `noteOff()` függvény működése ellentétes, ugyanis az éppen befejezendő hang példány változóinak „üresbe állítását” végezzük, így lehetővé téve később aktív hangnak ezen indexű hely használatát a `hang[]` vektorban.

```
void VstXSynth::noteOff (VstInt32 note, VstInt32 delta)
{

        for(int f_i=0;f_i<MAX_POLIFONIA;f_i++)
        {
                if (hang[f_i].note == note)
                {
                        hang[f_i].note=0;
                        hang[f_i].isOn=false;
                        hang[f_i].deltaki=delta;
                        hang[f_i].velocity=0;
                        hang[f_i].deltabe=0;
                        hang[f_i].fPhase1 = 0.f;
                }
```

```
}
```

Működése egyszerű. Egy for ciklus kikeresi azt a hangot, amelynek elnémítására parancs érkezett[21], és kiinduló értékekre állítja a változóit, majd az `isOn` flaget.

3.3.3. Az initProcess függvény

A MIDI kezelés ismertetése után a többi függvény is bemutatásra kerül, kiemelvén az `initProcess () és processReplacing()` függvényt, amik közül előbbi a harmonikusok és változók kezdeti értékadásáért felel. A dolgozat második részében ismertetett tranziensek és zaj modellezéséért felelős szűrőparaméterek forrásáról még nem volt szó. Ezeket *.txt* fájlok tárolják a **C://text/** könyvtárban, és minden hanghoz külön fájl tartozik. Ennek beolvasása is az `initProcess()`-ben történik, de mivel a tranziens-burkolók és zajgenerátorok a `processEvents` függvénnyel szinkronban hívódó `processReplacing()` függvénybe lettek beillesztve, természetesen ott kerülnek felhasználásra. A fájlkezelésről nagyon fontos megemlíteni, hogy valós idejű működésnél a fájlok fokozatos merevlemezről olvasása a hosszú olvasási idők miatt kivitelezhetetlen. Ezen probléma megoldására az összes fájl beolvasásra kerül az `initProcess()` meghívásakor, elkerülve ezzel a beolvasási problémákat. A mai számítógépek memória kapacitásának köszönhetően, ez már nem jelenthet problémát.

Az `initProcess()` az inicializálásokért felel, és ebből következően a host által csak egyszer, a program elején hívódik meg. Először tekintsük át a függvény első főbb részét, a fájlkezelés lényegét, ami az `initProcess()`-ben lett implementálva. A fájlkezelés alapja egy olyan struktúra, ami a következő változókat és mutatókat tartalmazzák.

```
typedef struct knote
{
	double snote;			// A MIDI-kód értéke
	int numharm;			//a harmonikusok száma
	double norm_freq;		//a névleges frekvencia értéke
	double real_freq;		//a valós frekvencia értéke
	double* harm_stat1;		//a harmonikusok amplitódóinak vektora
	int numzaj;			//az alkalmazott zajszűrők száma
	double* ad;			//a zajszűrő paramétereire mutat(nevező e.h.)
	double* bd;			//a zajszűrő paramétereire mutat(számláló e.h.)
	double* a_tr_a;			//felfutó tranziens szűrő paraméterek(nevező)
	double* b_tr_a;			// felfutó tranziens szűrő paraméterek(száml.)
	double* a_tr_d;			//lecsengő tranziens szűrő paraméterek(nev.)
	double* b_tr_d;			//lecsengő tranziens szűrő paraméterek(száml.)

} s_note;
static s_note notes[128];
for(int ix=0; ix<128; ix++)
	{
	sprintf(s, "c:\\text\\%d.txt", ix);
```

A `static s_note notes[128]` egy 129 példányt tartalmazó vektor hoz létre. Ennyi különböző hang megkülönböztetésére képes a program, ebből következően a helyes működéshez pontosan ennyi *.txt* fájl tárolására, és egyszeri beolvasására van szükség.

[21] Vegyük észre, hogy note off státuszú MIDI üzenetnél is felhasználódik (ahogy a függvény paraméterlistájában is látszik) a note és delta értéke.

A beolvasás for ciklussal történik, ami 0-128-ig számozott *.txt* fájlokat egymás után beolvassa a memóriába. Az algoritmus megtalálható a CD mellékleten, a dolgozatban nem kerül ismertetésre (Elég annyit ismernünk, hogy a továbbiakban a `notes[index].valtozo` -val tudunk a megfelelő lenyomott hang paramétereire hivatkozni).

A függvény második fő részében a korábban ismertetett hang struktúra tagjainak kezdeti értékadása történik. Ami itt talán a legfontosabb rész, hogy a `hang` vektor összes példányának az olyan változói, mint (`velocity, deltabe, deltaki, fPhase1`) nullába állítódnak, valamint az `isOn flag=false` értéket kap:

```
for(int i=0;i<MAX_POLIFONIA;i++)
{.
.
.
.
hang[i].note=0;
hang[i].velocity=0;
hang[i].deltabe=0;
hang[i].deltaki=0;
hang[i].isOn=false;
hang[i].fPhase1  = 0.f;
.
.
.}
```

A harmadik részben a korábban is említett hullámalakok létrehozása a cél. A modellezéshez felhasznált *text* fájlokat átnézve kiderült, hogy a megvalósítani kívánt hangszer sípjai jellemzői közé tartozik, hogy legalább három, és legfeljebb 26 harmonikust tartalmaz. Mivel minden hanghoz változó számú harmonikust kell generálnunk, ez legegyszerűbben úgy kivitelezhető (és a kapacitásaink is megengedik), hogy előre létrehozzuk az összes lehetségesen előforduló harmonikus komponenst egységnyi amplitúdóval, és később csak annyi kerül kiválasztásra, amennyire szükség van, majd ezeket a megadott amplitúdóval súlyozzuk.[22]

```
sinus0[i] =  (sin(2. * pi * 1 *i/ kWaveSize));
sinus1[i] =  (sin(2. * pi * 2 *i / kWaveSize));
.
.
sinus24[i] =  (sin(2. * pi * 25 *i / kWaveSize));
sinus25[i] =  (sin(2. * pi * 27 *i / kWaveSize));
```

Az utolsó fontos esemény a már korábban említett frekvenciatábla feltöltése. A frekvenciatábla egy olyan vektor, amelyben a leütött hang frekvenciái kerülnek eltárolásra. Az `a` és `k` konstansok és egy `for` ciklus segítségével a következő módon feltöltünk egy frekvenciatáblát, amely az összes MIDI-eseményre tartalmazni fog egy frekvenciaértéket, ami a .txt fájlból kiolvasott értékekkel megegyező.

```
for (VstInt32 iz = 0; iz < kNumFrequencies; iz++)
{
      freqtab[iz] = notes[iz].real_freq;
```

[22] Ez utóbbira csak a processReplacing()-ben kerül sor, az initProcess()-ben az egységnyi amplitúdójú „oszcillátorok” létrehozása történik.

```
}
```

Az összes MIDI-esemény a `kNumFrequencies` értékével azonos, amely szintén a kód elején nyert értéket, így végeredményben a szintetizátor 128 különböző lenyomott billentyűre érzékeny.

3.3.4. A processReplacing függvény

A `processReplacing()` ismertetése előtt néhány kisebb függvény következik: A `setSampleRate` és a `setBlockSize()`.[15]

```
void VstXSynth::setSampleRate (float sampleRate)
```

Ez a függvény a legmagasabb frekvenciájú hang szolgáltatásáért felelős, ugyanis ez erősen függ a mintavételi frekvenciától, aminek értéke általános hangkártyákra 44100 Hz. Ennek függvényében az `fScaler` paraméter nulla és egy közötti értéket kap, amely a lenyomott hang alapfrekvenciájának a változtatásánál használódik fel.

```
fScaler = (float)((double)kWaveSize / (double)sampleRate);

void VstXSynth::setBlockSize (VstInt32 blockSize)
```

A `setBlockSize` függvény pedig azt a maximum blokkméretet állítja be, amelyet a `processReplacing()` függvény egyszerre kezelni tud. A hangszer a maximális ilyen méretű blokkra törekszik.

Az elkövetkezendő részek átlátása megköveteli az eddig tárgyalt összes rész elsajátítását, a kimeneti minták generálása ugyanis a `processReplacing()` feladata, amelyben felhasználódik az eddig tárgyalt mindkét struktúra szerkezet (*`hang`*, *`notes`*, amik közül látható volt, hogy az előbbi felelős a polifónikus megszólalásért, az utóbbi pedig a megfelelő megnyitott fájl adatainak a felhasználásáért). A másik fontos – ami eddig többször hangsúlyozandó információ volt (processEvents résznél) – a `processReplacing()` és `processEvents()` szoros egymás után nagy gyakorisággal meghívódó, egymással szinkronban működő függvények kapcsolata.

```
void VstXSynth::processReplacing (float** inputs, float** outputs, VstInt32
sampleFrames)
```

Lényegében ez a függvény az audio adatfolyam „processzálásáért” felel, és az egész valós idejű folyamat ebben a függvényben játszódik le. A blokk alapú feldolgozáshoz kapcsolódóan érdemes tudni, hogy a host program a feldolgozandó blokk méretétől függően többször meghívja ezt a függvényt, ami feltételezhetően több mint 10 meghívást jelenthet másodpercenként.[23]

Tekintsük át a paraméterlistát. Amikor meghívásra kerül a függvény, kimeneti és bemeneti többszörös mutatókat vesz, amely kimeneti sztereó jel deklarását szolgálja a következőképpen. Működés közben mindkét kimenetre ugyanazon értékeket adjuk ki, ami

[23] Tipikus hiba mikor, kinullázunk olyan puffer, vagy kimeneti változókhoz kapcsolódó értékeket, ami nem teszi lehetővé a folyamatos kimeneti hallható hangsávot, hanem „kattogást” okoz. Ilyen két kattogás közötti idő a processReplacing két meghívása közötti idő.

később látható lesz. A harmadik átvett paraméter a blokk mérete, amit `sampleFrames` névvel veszünk át és a továbbiakban `frames` néven is használunk.

```
float* out1 = outputs[0];
float* out2 = outputs[1];

memset (out1, 0, sampleFrames * sizeof (float));
memset (out2, 0, sampleFrames * sizeof (float));

VstInt32 frames;
out1 += sampleFrames;
out2 += sampleFrames;
```

Ezután a `memset` függvények a kimeneti puffernek nulla értéket adnak, és a blokk (`sampleFrames`) végére állítják a kimeneti puffer mutatóját. Erre azért van szükség, mert ha a függvényünk nem „észlel ” lenyomott billentyűt, lefutása után 0 értéket kell adjon a kimenetére. Ha nincs aktív leütött billentyű, akkor a program nem is tér rá a következő részekre, amely aktív működésekor visszaállítja a kimeneti puffer mutatóját, majd felülírja az előző 0 kimeneteket. A fenti kód ezért nevezhető a kései `if` korai `else` ágának, mert az `if` megtörténtekor az `else` ágat „semlegesíti”.

Ezt lehet nyomon követni a következő for ciklusra, és `if` ágra való továbbhaladással. A `for` ciklus a polifóniát valósítja meg. Emlékezhetünk a `noteOn` függvényünk szerkezetére, ami a rendezést követően a következő üres helyre helyezte a legutóbb lenyomott hangot, amelyet az `n_i` index jelölt, vagyis az `n_i` index tartalmazza, az éppen egymással egy időben leütött hangok sorszámát, így az ezeket tároló `hang[h_ind]` vektoron csak annyiszor fut végig a `for` ciklus, ameddig szükséges. Az `if` szerkezet ugyanis a hang megszólalásának a feltételét vizsgálja (ha `isOn == true` és `velocity` nem 0 értékű), aminek teljesülésével felépül a későbbiekben a hang. Természetesen ekkor a kimeneti puffer mutatóját visszaállítjuk a blokk elejére, különben a fenti `memset` függvény miatt csak nullák jelennének meg a kimeneten. A `for` ciklus és az if feltételvizsgálat:

```
for(int h_ind=0; h_ind < n_i+1; h_ind++)
{
            frames = sampleFrames;

if ((hang[h_ind].isOn == true) && (hang[h_ind].velocity != 0))
      {
            out1 -=sampleFrames;
            out2 -=sampleFrames;
```

Az előbbi feltételvizsgálatnak eleget téve engedélyezve van a hang megszólalása, a `for` ciklus pedig annyi ilyen hangot engedélyez, amennyi párhuzamosan megjelenik. A hang felépüléséhez meghatározandó megfelelő maszkolással a MIDI-kódja (`melyik`), a kód által a frekvenciatáblából kinyert alapharmonikusának frekvenciája (`baseFreq`), valamint a lehetséges összes harmonikusra deklarált módon, mutatókkal fogunk a későbbiekben hivatkozni. Az `fScaler` a mintavételi frekvencia alapján korrigálja a kívánt frekvenciát. [15]

```
int melyik = hang[h_ind].note & 0x7f;
float baseFreq = freqtab[melyik] * fScaler;
float* wave0 =sinus0;
float* wave1 =sinus1;……;
float* wave25 =sinus25;
```

Mivel várhatóan a leütött billentyű a feldolgozott blokk belsejében történt, a blokk leütés előtti részeivel megegyező méretű kimeneti blokk kinullázódik, majd az ezt követő részekre engedélyezhető tényleges kimenet. Ezt a pozíciót jelöli a `deltabe` érték, amely a `processEvents()` függvényben (azon belül is a `NoteOn()`-ban) került értékadásra. A kód kiterjeszti az eseményhalmazt arra az esetre is, ha a `deltabe` nagyobb magánál a blokknál, de ideális körülmények között ez soha nem történhet meg.

```
if (hang[h_ind].deltabe > 0)
{
      if (hang[h_ind].deltabe >= frames)
      {
            hang[h_ind].deltabe -= frames;
            return;
      }

      out1 += hang[h_ind].deltabe;
      out2 += hang[h_ind].deltabe;
      frames -= hang[h_ind].deltabe;
      hang[h_ind].deltabe = 0;
}
```

A blokk pozícionálása rész után nem maradt más hátra, mint a blokk azon ún. „aktív” részének vizsgálata, amely a fentiek alapján a leütött hangra fog kimenetet produkálni. Ez a feldolgozás folyamata. A feldolgozás mintánként történik, amelyet egy `while` ciklus valósít meg, amely addig kerül lefutásra, amíg a blokk végére nem ér. Ez a következőképpen néz ki,

```
      while (--frames >= 0)
{
```

amelynek a feladata röviden mintánkénti orgona hangjának létrehozása a leütött hang szerint. A generálás a fájlok által tárolt modell alapján történik, ezért szükséges a zajminták, a tranziensek, az alap- és felharmonikusok mintáinak előállítása. Az algoritmus az additív zajgenerátorral folytatódik, amelyet megelőz a zajszűrő számláló és nevező együtthatókat átmenetileg tároló vektorának deklarálása.

```
      double den_d[3];
      double num_d[3];
```

A sípzaj generálása az elmélet szerint annyi szűrőt igényel, ahány zajcsúcs található a leütött hang zajspektrumában. Ezt a célt szolgálja a `for` ciklus, amely viszont annyiszor fut le, ahány zajcsúcs van a hangban, ismételten felülírva a `num_d, den_d` értékét, vagyis a `for` minden lefutáskor a következő zajszűrő aktivizálódik. A szűrőre vezetett bemeneti Gauss-zaj mintái a `hang[h_ind].noise[hang[h_ind].g]` vektorban tárolódnak. ‚vagyis minden hanghoz külön fehérzaj vektor tartozik, melynek futóváltozója a `hang[h_ind].g` (azaz a hanghoz tartozó g)[24]. Ennek az értéke a fájlkezelés, és beolvasás folyamatában megtörtént, így azoknak felhasználásával a kód a következő[25]:

[24] A `g` értéke – az `if()` ág végén látható módon – 0-2-ig egyesével nő minden `while` ciklusban, ami 2 után újra 0-ba állítódik.

[25] A jobb átláthatóság kedvéért a `hang[h_ind] N-el` lett jelölve. A 2.3.4. fejezetben lévő egyenlet alapján történik a zajminták generálása.

```
for (int m = 0; m < notes[melyik].numzaj*3; m++,m++,m++)
{
                    den_d[0]=notes[melyik].ad[m];
                    den_d[1]=notes[melyik].ad[m+1];
                    den_d[2]=notes[melyik].ad[m+2];
                    num_d[0]=notes[melyik].bd[m];
                    num_d[1]=notes[melyik].bd[m+1];
                    num_d[2]=notes[melyik].bd[m+2];
.[26]
.
N.feedbuff[N.g] =(1/den_d[0])*
(num_d[0]*N.noise[N.g]+num_d[1]*N.noise[N.g-1]+num_d[2]*N.noise[N.g2+Size]-
den_d[1]*N.feedbuff[N.g-1]-den_d[2]*N.feedbuff[N.g-2+Size]);
.
.
            N.sum[N.g] = N.sum[N.g] + N.feedbuff[N.g];
}
```

A zajminták láthatóan a hangonkénti `feedbuff` vektorban tárolódnak (majd felülíródnak), melyek értékeit lefutásonként a `sum` vektor gyűjti. Természetesen ez az összeadás annyiszor történik meg egy mintára (a `while` egy lefutására), ahány zajszűrő szükséges a sípzaj előállításához.

A továbbiakban minden komponensre (harmonikusokra és zajra) történő minták előállítása viszonylag nagy terjedelmű, de az áttekinthetőség és megérthetőség szempontjából egyes komponensek kiemelése volt csak indokolt. Ezért a következőkben csak néhány komponenshez tartozó algoritmus bemutatása történik.

A tranziens burkológörbék előállítása a szűrők használata miatt hasonló elveken alapszik. A hasonlóság és nagy terjedelem miatt csak a felfutó tranziensek dokumentációjára került sor.

Három fő részre bontható a következő algoritmus. Az első részben a fájlból, az inicializálásnál kinyert adatokat kell megfelelő tárolókba (jelen esetben vektorokba) másolni a helyes használat érdekében, mert az IIR-szűrő együtthatói egy vektoros adatszerkezetben vannak, a használathoz pedig mindenképpen külön vektorokra van szükség elkülönítve a különböző harmonikusok paramétereit, elkülönítve még a számláló, nevező együtthatóit is. Ez az algoritmus fizikai tartalma szempontjából kevésbé fontos, használati szempontból viszont elengedhetetlen, ezért a lényegi része dokumentálásra került.

```
for (int h = 0,k = 4; h < 4; h++)
      {
      atra1[h] =notes[melyik].a_tr_a[h];
      atra2[h] =notes[melyik].a_tr_a[h+k];
      atra3[h] =notes[melyik].a_tr_a[h+2*k];
      if(notes[melyik].numharm > 2)[27]
      {atra4[h] =notes[melyik].a_tr_a[h+3*k];}
      .
      .
```

[26] Az `if else` szerkezet (amely csak a beillesztett kódból lett kihagyva a kevésbé fontos fizikai tartalma miatt) a megvalósítási módszer miatt szükséges, használata más módszereknél elkerülhető. Ez legegyszerűbben úgy értelmezhető, hogy a negatív indexelés elkerülése céljából lett megvalósítva.

[27] A kód megírásakor figyelemmel kellett tartani azt a fontos tényezőt, hogy ha több mint 2 harmonikus van az 4 burkológörbe generálását jelenti, mert 3 harmonikus mellett a zajnak is előállítjuk a burkolóját.

```
.
if(notes[melyik].numharm > 25)
{atra27[h] =notes[melyik].a_tr_a[h+26*k];}
```

Az adathalmaz kinyerését egy for ciklus végzi. `atra1[h]` vektor az 1. harmonikushoz (alapharmonikushoz) tartozó tranziens szűrő számláló együtthatóinak halmaza (ebből következően a nevezőit pedig `btra1[h]` tárolja, `a btrd1[],atrd1[]` vektorok pedig a lecsengő tranziensekhez tartoznak), amely 4 elemű tárolását végzi (a harmadfokú IIR-szűrő együtthatóinak száma miatt). Az `if` ágak azt hivatottak eldönteni mennyi harmonikus van, ebből következően hány ilyen vektort kell feltölteni.

A második fő részben a burkoló mintáinak a generálása történik. A tranziens részek tárgyalását végig függővé válik a `notes[melyik].numharm` változótól, vagyis a harmonikusok számától, így a lefutáskor a host folyamatosan „`numharm`” feltételvizsgálatokon megy keresztül.

```
if(hang[h_ind].flag_128 == 0 )
{
      hang[h_ind].y_old1=hang[h_ind].y_1;
      hang[h_ind].y_old2=hang[h_ind].y_2;
      hang[h_ind].y_old3=hang[h_ind].y_3;

      if(notes[melyik].numharm > 2)
      {hang[h_ind].y_old4=hang[h_ind].y_4;}
```

A `hang[h_ind].flag_128` változó egy olyan hang struktúrájában lévő változó, amely célja az interpoláció miatti késleltetett számolás. Mivel valós időben történik a burkológörbe minták számolása is, és a 2.3.3. fejezetben leírtak szerint a két generált minta érték közötti 128 pontos interpoláció számolása megkövetelt, az eredeti minták kiszámolása elegendő minden 128. minta értékre (a while minden 128. lefutására), a többi köztes időpontban nem lépünk be a számolás részbe, hanem az előzőleg kiszámolt két minta közötti interpoláció számolása hajtódik végre.[28] A következőkben a 3. és 4. harmonikus aktuális mintái a következőképpen számolandó:

```
.
.
.
N.y_3= btra3[0]*N.h0+btra3[1]*N.h1+btra3[2]*N.h2+btra3[3]*N.h3-
atra3[1]*N.y1_3-atra3[2]*N.y2_3- atra3[3]*N.y3_3;

if(notes[melyik].numharm > 2)

{N.y_4= btra4[0]*N.h0+btra4[1]*N.h1+btra4[2]*N.h2+btra4[3]*N.h3-
atra4[1]*N.y1_4-atra4[2]*N.y2_4- atra4[3]*N.y3_4;}
```

A `h0, h1, h3` az egységugrás bemenet értékeit, a `btra3[], btra4[], atra3[], atra4[]` a megfelelő együtthatók értékeit tárolja. (Az N ismét a `hang[h_ind]` rövidítését jelenti az áttekinthetőség érdekében). Az `y_1….y_26` az egyes harmonikusokhoz, valamint zajhoz tartozó interpolálás előtti burkológörbe mintáit tartalmazzák, az `y_old1, y_old2… y_old26` pedig az éppen aktuális érték eltárolását végzi (ennek értékadása a fenti kódrészletben látható első négy komponensre), ami a következő minta kiszámolásánál az előző minta értéket jelenti.

[28] Az első lefutáskor belépünk az if ágba, de a lefutás után csak egy előző minta létezik még. Ekkor az interpoláció természetesen a 0, és ezen minta értéke között számolódik 128 különböző pontban.

Ez tulajdonképpen az interpolálásnál kerül felhasználásra (mivel ott a két bemenetnek az aktuális és előző minta értéke tekinthető). Az interpolálást végző algoritmus részlete a következő

```
N.yip_out3 = N.y_3 + N.flag_128*(N.y_3 - N.y_old3)/(128);

if(notes[melyik].numharm > 2)
{N.yip_out4 = N.y_4 + N.flag_128*(N.y_4 - N.y_old4)/(128);}
```

, ahol szintén a 3. és 4. komponensre való számolást lett beillesztve. Itt látható igazán az `y_3`, és `y_old3` közötti részek „feltöltése”, ugyanis a már ismert `N.flag_128` 0-127-ig futó változó biztosítja az előző, és az aktuális minta értéke közötti minták kiszámolását, amit a `N.yip_out3` (`hang[h_ind].yip_out3`) változóban tárolunk el.

Ezzel megkaptuk az aktuálisan billentett hang tranziens burkoló mintáit is, de mielőtt a kimeneti puffer értékeinek összeállítására rátérnénk, egy egyszerűbb feltételvizsgálattal meghatározandó, hogy melyik változó tartalmazza a generált zaj burkolómintáinak értékét. Ez természetesen a harmonikusok számának függvénye:

```
if(notes[melyik].numharm == 2){N.zajip_out=N.yip_out3;}
else if(notes[melyik].numharm == 3){N.zajip_out=N.yip_out4;}
else if(notes[melyik].numharm == 4){N.zajip_out=N.yip_out5;}
.
.
.
else if(notes[melyik].numharm == 24){N.zajip_out=N.yip_out25;}
else {N.zajip_out=N.yip_out26;}
```

, és az így nyert `N.yip_out` értékét adjuk a `N.zajip_out` változónak.

Ebben a szakaszban az orgona hangjának modell alapú előállításának mintájához minden komponens előállt, így a kimeneti pufferben az alap- és felharmonikust, majd a zajt megszorozzuk a burkológörbe adott mintájával, majd összegezzük őket. A kimeneti puffer mutatóját ezután a puffer következő elemére állítjuk (`out*++`), majd a `while(--frames)` addig fut le újra, amíg a blokk végére nem ér. A kimeneti minta a programkód szerint

```
*out1 =
(notes[melyik].harm_stat1[0] *wave0[(VstInt32)N.fPhase1 & mask]* N.yip_out1
+ notes[melyik].harm_stat1[1]*wave1[(VstInt32)N.fPhase1 & mask]* N.yip_out2
+ N.sum[N.g]* N.zajip_out)* fVolum

if(notes[melyik].numharm > 2)
{*out1 +=
 fVolume1*notes[melyik].harm_stat1[2] *wave2[(VstInt32)N.fPhase1 & mask] *
N.yip_out3 ;}

if(notes[melyik].numharm > 3)
{*out1 +=
fVolume1*notes[melyik].harm_stat1[3] *wave3[(VstInt32)N.fPhase1 & mask] *
N.yip_out4;}
.
.
.
if(notes[melyik].numharm > 25)
```

```
{*out1 +=
fVolume1*notes[melyik].harm_stat1[25] *wave25[(VstInt32)N.fPhase1 & mask] *
N.yip_out26 ;}

*out2++) += *out1++;
```

, amiben a harmonikusokra hívatkozás az erre deklarált `*wave0, *wave1, *wave2…., *wave` mutatókkal történik, amelyek a `notes[melyik].harm_stat1[]` vektor által tárolt amplitúdókkal megfelelően súlyozódnak. Az `N.fPhase1` a megfelelő fázisból indulást jelenti, aminek folyamatos frissítése a zavartalan folytonos hang kiadásának feltétele.[29]
Az if szerkezetek az eddigi szerepük szerint működnek. Annyi harmonikust engedélyez, amennyihez szükséges zaj és traziens burkoló minta is előállt, és természetesen csak pontosan ennyi számú harmonikust tud a megfelelő amplitúdóval megszorozni. Végső lépésként a második kimeneti pufferbe is beletesszük az első pufferben tárolt mintát, és mindkét puffer mutatóját léptetjük.

Egy `while(--frames)` teljes végigfutása alatt természetesen előáll az engedélyezett hangunk egy blokkja, de a `processReplacing()` legelején tárgyalt for ciklus [30] biztosítja azt, hogy ez a folyamat annyiszor végre hajtódjon – annyi engedélyezett hangra -, amelynek a párhuzamos megszólaltatását kívánjuk. Ez tehát a polifónia alapja. A polifónikus folyamatos működés feltétele volt, hogy miután elengedtünk egy hangot, a tranziens számolásnál használt minden komponens segédváltozóját az inicializáláskor beállított értékre állítsuk vissza, tehát kinullázzuk. Ezeknek a kezdeti értékre állítása nélkül a tranziens burkolók hallhatóan működnek ugyan, de csak a programban betöltése után közvetlenül az első engedélyezett hangra kizárólag. Ennek oka, hogy mivel nem nulláztuk ki a változókat (hangonként), a tranziens burkológörbék működtek tovább, folyamatosan egységnyi értéket generálva a kimenetükön, vagyis a nullából való felépülésük nem tudott újra elkezdődni.

A másik fontos dolog ami a polifónia megvalósításkor történt, a zaj (`hang[h_ind].g, hang[h_ind].sum[], hang[h_ind].feedbuff[]`), tranziens (`y_1…y_26, y1_1…y1_26, y2_1…y2_26, y3_1…y3_26, h1, h2, h3`) segédváltozóinak adott hanghoz rendelése (struktúrában inicializálása). Ezeknek a nulla értékadása a `processReplacing()`-ben megtörténik, de megvalósíthatása a `noteOff()` függvényben is lehetséges.

[29] Tipikus hiba szokott lenni a fázisok elhanyagolása. Ilyenkor minden a `processReplacing()` meghívásából adódan a blokkok összeillesztése nem lesz zavartalan, amely zavaró kattogásban nyilvánul meg.
[30] (`for(int h_ind=0; h_ind < n_i+1; h_ind++)`)

4.Tesztelés

A szoftvertesztelés szorosan illeszkedik a szoftverfejlesztés folyamatába. Célja a szoftverhibák felderítése, esetlegesen a szoftverminőség mérése, a program futtatása mellett.

A működéshez szükséges a paramétereket tartalmazó text fájlok **C:\\text** mappába másolása. A text fájlok 0-127-ig vannak elnevezve, egy text fájl elrendezése adott, melyet MATLAB segítségével egy példán keresztül a következőképpen generálunk le: 89-es MIDI-kódú text fájl létrehozásához a 89.mat fájl betöltése szükséges, amelyet a

save 89.txt midinote num_harm nom_freq real_freq harm_stat a_tr_a b_tr_a a_tr_d b_tr_d ad bd n_av noise_level -tabs –ASCII

paranccsal hoztam létre az analízis Császár községben végzett mérések adatait tartalmazó *.mat* fáljaiból.[1] Az orgonaszintetizátor Bourdon és Principál regiszterek sípjait modellezi.

A grafikus felületet mutató 16. ábrán lévő csúszkák közül a második és harmadik aktív. Az előbbivel kis frekvenciakorrekciót lehet létrehozni 1 Hz-es határon belül, az utóbbival a kimeneti hangerőt lehet állítani. A hullámformák változtatását inaktívra állítottam, orgonasípok modellezésénél nem tulajdonítottam jelentőséget neki.

A program tesztelésének idején a lecsengést biztosító tranziens burkológörbéket megvalósító rész futási hibák miatt kikerült az algoritmusból, ezért elengedéskor kisebb pukkanások jelentkeznek.

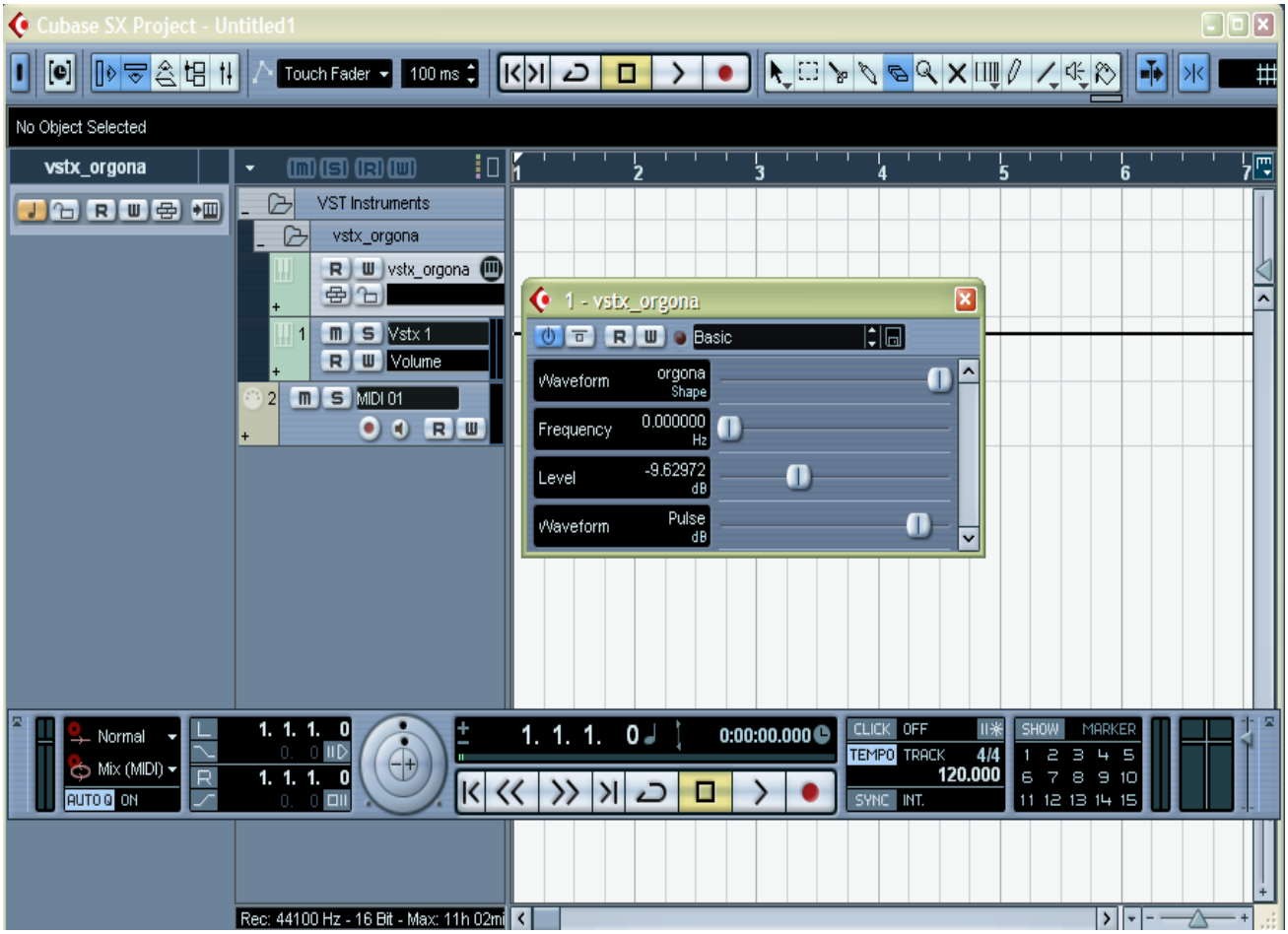
Monofónikus esetben a szintetizátor viszonylag jól működött, kisebb darabok lejátszására megfelelő, a lenyomott billentyűk alapján történik a hang keletkezése. A harmonikusokban gazdag hangok teltebben szóltak, mint a kezdeti fázisban lejátszott szimpla szinuszok. A felépülő tranziensek jól működnek, pukkanás ebben az esetben nincs, a zajjelenség meglétét pedig 40 dB-es hangerőnöveléssel ellenőriztem a fejlesztés ideje alatt. A zajgenerátorok kifogástalan működését bizonyítják az egymás utáni hangok közötti zaj mind a karakterisztikájában, mind hangerőben való eltérés. Jól megfigyelhető azonban a mély hangok késleltetett tranziensei a magas hangokhoz képest.

Kattogásból eredő hibákat nem észleltem. Ez azt jelenti, hogy a hangok zaj-, tranziens, és egyéb paraméterei alaphelyzetbe csak a program futása elején, valamint a hang befejeződésekor kerülnek. Két gyakori lefutást végző `processReplacing()` között nem nullázódnak ki ezek a segédváltozók. Véleményem szerint a meglévő hibákért nem a polifónikus MIDI kezelést végző függvények felelősek. A korábbi szakaszokban produkált ugyan olyan hibát, hogy nem megfelelő hangot szólaltatott meg a plugin, hanem egy korábbban lenyomott billentyű „maradt benne” a hang változóit tároló `hang[]` tömbben, de ez kijavításra került.

Negatív tapasztalatnak könyvelhető el, hogy kitartott hangoknál 4-5 másodperc elteltével a hangok halvány elhajlása következett be, hamissá vált az adott hang. Polifónikus esetben az elhajlás szembetűnőbb. Jelenleg 6 különböző billentyű lenyomására érzékeny, de ha az összes lehetséges billentyűt lenyomjuk, a rendszert túlvezéreljük, ez kijavításra vár. Lehetéges, hogy a felharmonikusok miatti nagyszámú szinuszgenerátor egyidőben történő minták összege túllépi a megengedett határértéket. Az algoritmus alapján a 7. billentyű lenyomása egy korábbi megszüntetését jelenti. A polifónia számának azonban nincs

jelentősége, mert könnyen megváltoztatható. MIDI fájl vezérlésével az orgona kattogása szembetűnőbb, lassabb szólamok élvezhetően kiadják a hangszer hangját.

Az ábrán látható az orgona szintetizátor Cubase SX3-ba betöltött képe látható. A program fejlesztése vst.sdk 2.4, tesztelése Cubase SX3 3.1.1-es verzióval történt.



16. ábra. A VST_orgona grafikus felülete

5.Összefoglalás

5.1. Eredmények

A kiírásban szereplő feladatokat – az előzetesen megfogalmazott célok szem előtt tartásával – megoldottam, melynek továbbfejlesztési lehetőségeiről a következő alfejezetben térek ki.

Az orgona digitális jelmodellje alapján történő szintézis módszerek segítségével létrehoztam MATLAB környezetben orgona hangját előállító algoritmust, amelynek során végigkövethettem a szintézis különböző folyamatait, és ellenőrizhettem azok működését, felhasználva a VST környezetben történő implementálásra.

A modellben megvalósult a hang harmonikusainak, felfutó és lefutó tranzienseinek, valamint a jellegzetes sípzajának megvalósítása. A CD-mellékleten is megtalálható hanganyag a lecsengési tranziense implementálásával és anélkül. Az előbbivel a T értékének függvényében változtatható a legenerált kimenet és a wav hangminta hossza.

A másik lényeges eredmény a VST-ben megírt orgona, amely polifónikus működésű, implementálva a harmonikusokat, zajt és felfutó tranzienseket egyaránt. Megvalósítottam a text fájlok beolvasását, a polifónikus MIDI-kezelést, és a polifónikus kezelésből adódó független hang- és annak komponensei létrehozásának billentésenkénti kezelését.

A félév során betekintést nyertem az orgona hangjának fizikai és pszichoakusztikai jellemzőket összefogó irodalmakba, megtapasztaltam a különböző szintézis módszereknek az előnyeit, hátrányait, valamint így végigkövethettem azt az utat, amelyen a szintézistechnika eljutott a fizikai alapú modellezéshez, majd az általam is felhasznált hangszer jelmodelljéhez.

A VST környezetben történő szintetizátor plugin olyan tapasztalatokkal gazdagított, amelyek során nemcsak hangszermodellezés, hanem általános audioeffekt fejlesztési készségeket is magamévá tehettem egy modern és nagy lehetőségeket nyújtó fejlesztői környezetben. A megírt szoftver alkalmas MIDI billentyűzet és MIDI fájlok általi vezérlésre egyaránt.

A digitális modell alapján véleményem szerint viszonylag kevés számítási időt vett igénybe, és fejlesztői környezetbe való implementálása is optimális viszonyok között megoldható. Összességében a jelmodell alapú, valós idejű szintézist lehetővé tevő módszer hatékonynak bizonyult.

5.2. Továbbfejlesztési lehetőségek

A projekt továbbfejlesztési lehetőségei közül kiemelhető a 4. fejezetben említett hibák kijavítása mellett grafikus felület tervezése, amely több funkció ellátására képes.

Egy ilyen funkció lehet például valamilyen vezérlőparaméterrel kapcsolható regisztertípusok közötti átkapcsolás. Ehhez kapcsolódóan szükséges paraméterfájlok előállítása, majd projekthez hozzáadása.

Másik fejlesztési lehetőség a külső körülményeket lehetővé tevő modell implementálása a rendszerbe. Ez zengetőalgoritmussal és pozícionálással valósítható meg mintegy templomi környezetet varázsolva a hangszer köré.

A hangszer hű modellezésen túllépően effektekkel lehet érdekesebbé, gazdagabbá tenni a szintetizátor hangzásvilágát. Különböző modulációkkal, hanghajlításokkal érdekes hangzást alakíthatunk ki.

Egyszerűsítési lehetőségként megemlíthető, hogy a további fejlesztés szempontjából célszerű lehet az algoritmus külön fájlokban történő szétbontása, új függvények létrehozásával.

6.Irodalomjegyzék

[1] Márkus János – Orgonasípok hangjának jelmodell alapú szintézise diplomaterv, Budapesti Műszaki- és Gazdaságtudományi Egyetem, Budapest 1999.

[2] Tarnóczy T., „Zenei akusztika” Zeneműkiadó, Budapest, 1982.

[3] Tarnóczy T., „Teremakusztika” Akadémiai kiadó, Budapest, 1986.

[4] Angster J., „Orgona ajaksípok megszólalásának és rezgésének korszerű mérései és eredményei” kandidátusi értekezés, MTA MMSz Akusztikai Kutatólaboratóriuma, Budapest, 1990.

[5] Fletcher, N.H., Rossing, T.D., „The Physics of Musical Instruments” Springler-Verlag, New York, 1991.

[6] Albert P., „Kísérlet az orgonahangzás elektronikus szintézisére I–IV.” Kép- és hangtechnika 1980/3. 75–84. o., 1985/5. 137–142. o., 1986/5. 141–148 o., 1987/2. 33–41. o.

[7] Eduardo Reck Miranda „Computer Syntheris for the Electronic Musician”, Focal Press, Oxford 1998.

[8] Papp Sándor Róbert , Hegedűhang digitális szintézise diplomaterv, Budapesti Műszaki- és Gazdaságtudományi Egyetem, Budapest, 2007.

[10] Fodor Gy., „Hálózatok és rendszerek analízise I.” egyetemi jegyzet (55014), Műegyetemi kiadó, Budapest, 1994.

[11] Fodor Gy., „Hálózatok és rendszerek analízise II.” egyetemi jegyzet (55014), Műegyetemi kiadó, Budapest, 1994.

[9] Péceli, G., „A common structure for recursive discrete transforms”, IEEE Transactions on Circuits and Systems, CAS-33., 1035–36. o., 1986.

[12] Tim Tully’s book MIDI For The Professional [http://www.tigoe.net/pcomp/code/serial-communication/midi]

[13] David Miles Huber„The MIDI Manual: A Practical Guide to MIDI in the Project Studio Edition3”, Focal Press, 2007, Kiadó: ISBN 0240807987, 9780240807980

[14] Virtual Studio Technology, http://namm.harmony-central.com/Product-news/VST-2.0.html

[15] VST, Software Development Kit, http://www.steinberg.net/en/company/3rd_party_developer/sdk_download_portal/vst_23_audio_plug_ins_sdk.html#input

[16] Virtual Studio Technology, Wikipedia, http://en.wikipedia.org/wiki/Virtual_Studio_Technology

[17] Hammond-orgona, http://www.answers.com/topic/hammond-organ-1

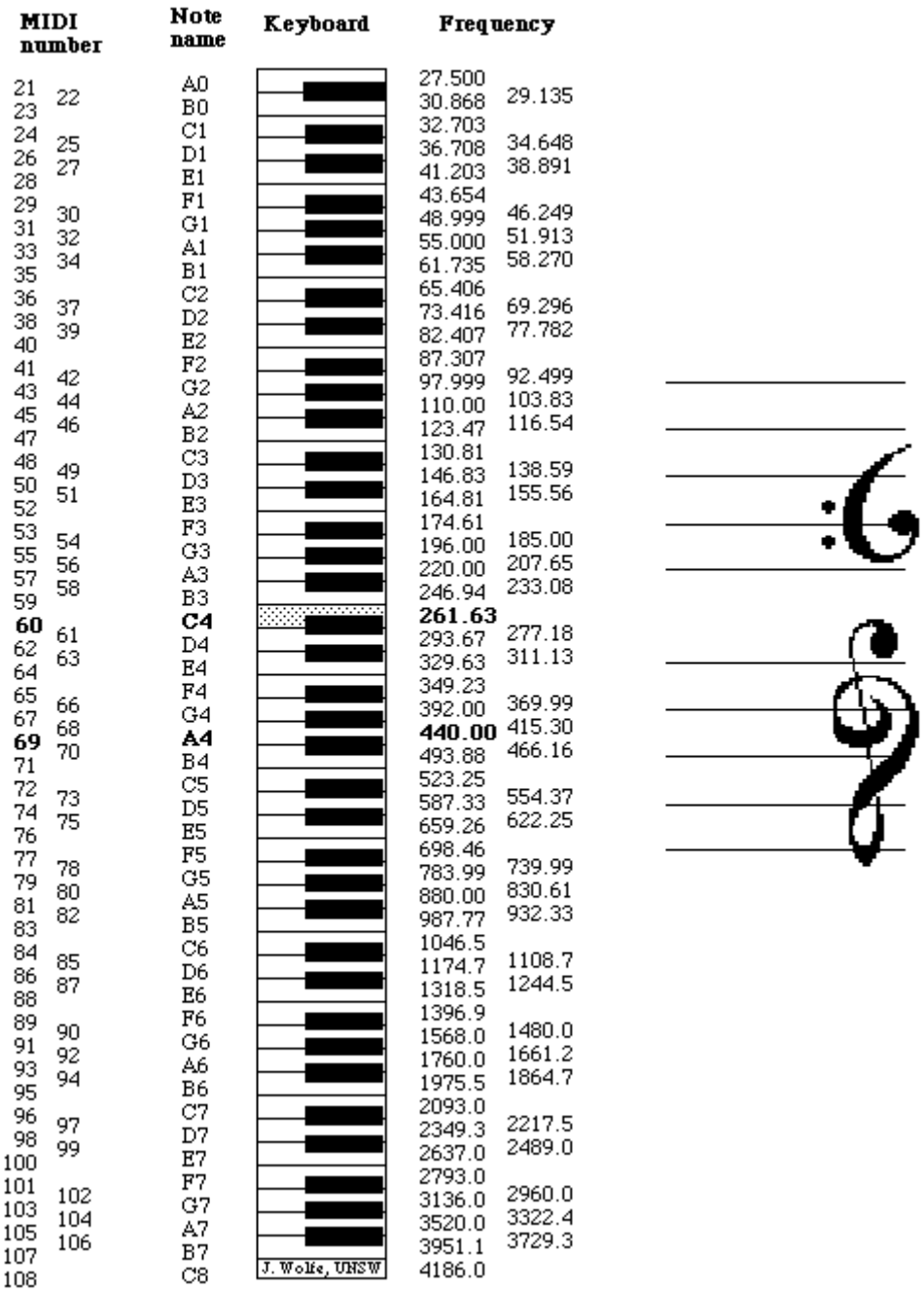
[18] Szubtraktív-szintézis http://www.abrudbanyay.hu/zifo/orgonatol.htm#A%20lebont%E1son%20alapul%F3%20szint%E9zis%20(Synthese%20soustractive)

[19] Garamvölgyi Zsolt, „Dobhang fizikai alapú szintézise” TDK dolgozat BME, Budapest 2006. http://home.mit.bme.hu/~garam/download/gzs_tdk.pdf

[20] Midinotes, http://www.dolmetsch.com/midinotes.gif

7.Függelék

F.1. MIDI-kódok



***17.* ábra: A MIDI-kódok billentyűk szerinti elhelyezkedése a klaviatúrán.[20]**

F.2. Steinberg VST SDK 2.4 könyvtárszerkezete [15]

artwork	VST Logo fájlok
bin	SDK Binaries
win	Windows platform eszközök
mac	MacOS X platform eszközök
doc	VST SDK Dokumentáció és licenc egyezmények
pluginterfaces	Interfész Fájlok
vst2.x	VST 2.x kezelői felületek
public.sdk	SDK forrásfájlok
samples	
vst2.x	VST 2.x Plug-in and Host samples
again	Gain mintaplugin
adelay	Delay mintaplugin
minihost	Minihost
vstxsynth	Egyszerű szintetizátor plugin
win	Windows platform fájlok (Win32 és x64, Visual C++ 2005-höz)
win.vc2003	hagyományos Windows platform fájlok (csak Win32, Visual C++ 2003-hoz)
win.vc6	Hagyományos Windows platform fájlok (csak Win32 ,Visual C++ 6.0-hoz)
mac	MacOS X platform fájlok (PPC és MacIntel arhitektúra, XCode 2.2-höz)
source	
vst2.x	VST 2.x bázisosztályok
vstgui.sf	VSTGUI (grafikus felület) Library
drawtest	
win	Visual C++ 2005 projekt fájl (Win32 és x64)
win.vc6	Visual C++ 6.0 projekt fájl (csak Win32)
mac	XCode 2.2 projekt fájl (PPC és MacIntel)
vstgui	VSTGUI forrájfájl (stabil 3.0 Sourceforge.net-ről)
Documentation	VSTGUI Dokumentáció

F.3. A MATLAB szintézishez szükséges adatok értelmezése

a_tr_a: Adott hang tranziens-felfutásához szükséges szűrő nevezőjének polinom-együtthatóinak (num_harm+1)*4-es mátrixa.

a_tr_d: Adott hang tranziens-lecsengéséhez szükséges szűrő nevezőjének polinom-együtthatóinak (num_harm+1)*4-es mátrixa.

ad: A zajszűrő nevezőjének polinom-együtthatóit tartalmazó n*3-as mátrix, ahol egy sor egy adott csúcs szűrőjének paraméterei.

b_tr_a: Adott hang tranziens-felfutásához szükséges szűrő számlálójának polinom-együtthatóinak (num_harm+1)*4-es mátrixa.

b_tr_d: Adott hang tranziens-lecsengéséhez szükséges szűrő számlálójának polinom-együtthatóinak (num_harm+1)*4- es mátrixa.

bd: A zajszűrő számlálójának polinom-együtthatóit tartalmazó n*3-as mátrix, ahol egy sor egy adott csúcs szűrőjének paraméterei.

Fs: mintavételi frekvencia, 44100.

harm_stat: A harmonikusok amplitúdójának értékeit tartalmazza.

Midinote: Az adott hang Midi-kódja.

n_av:

noise_level: Zajszint.

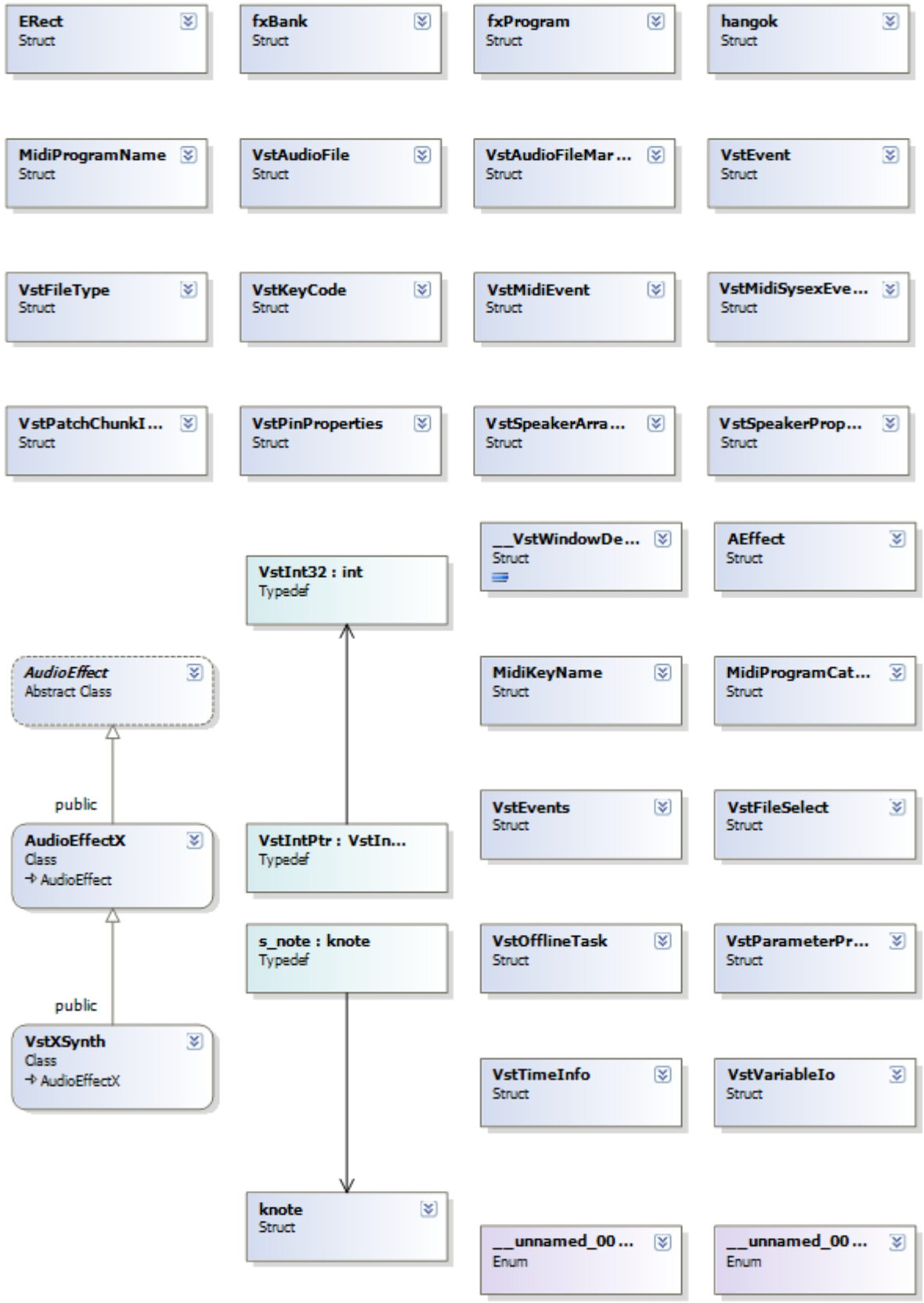
nom_freq: Az adott hang névleges frekvenciája.

num_harm: A harmonikusok száma.

real_freq : Az adott hang valós frekvenciája

Megjegyzés: A tranziens-együtthatók mátrixainak sora egy adott harmonikusra vonatkozik, az utolsó sor pedig mindig zaj tranziens paramétereit tartalmazza.

F.4. Egy VST szintetizátor részletes osztálydiagramja



F5. Fogalomtár

Percussion: Ütős hangszerek elnevezését jelenti, az elektronikus döbkeltés is ebbe kategóriába tartozik

Overdrive: A jel torzításán alapuló effekt.

Reverb: Késleltetésen alapuló effekt.

Phaser: Jellegzetesen hullámzó, huhogó hangot adó modulációs elven alapuló effekt.

Gain: A hang erősségét szabályozó primitív effekt.

Delay: Delay effekteknek a késleltetés-alapú effekteket nevezzük a zenei jelfeldolgozásban.

Note on: A hang aktiválását jelző MIDI információ.

Note off: A hang elnémítására érkező MIDI információ.

Pitch bend: A hangmagasság változtatását, hajlítását értjük alatta.

Velocity: A billentés erősségét meghatározó MIDI információ.

Plugin: Az általunk megírt VST alkalmazás lefuttatásával létrejön .dll formátumú fájlban, amelynek kezelését a host oldal (gazda program) végzi. A host program felelős a különböző függvények helyes sorrendű meghívásáért.
menzúra

dll: A DLL (Dinamic Link Library, szó szerint „dinamikus csatolású/hivatkozású könyvtár”) kifejezés a Windows operációs rendszerek alkalmazásainak (programjainak) segédfájljait, egészen pontosan az ún. megosztott könyvtárakat jelentik