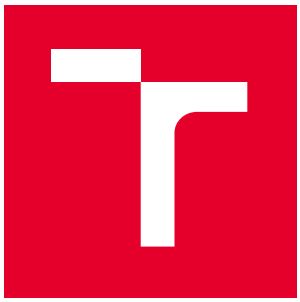


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

EXPERIMENTÁLNÍ SOFTWARE PRO DISTRIBUCI GRANULÁRNÍ SYNTÉZY DO PROSTOROVÉHO ZVUKU

EXPERIMENTAL GRANULAR SOFTWARE WITH SPATIAL SETTING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Tomáš Pospíšil**
AUTHOR

VEDOUCÍ PRÁCE — **MgA. Michal Indrák, Ph.D.**
SUPERVISOR

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Audio inženýrství**
specializace Zvuková produkce a nahrávání
Ústav telekomunikací

Student: Tomáš Pospíšil ***ID:*** 211807
Ročník: 3 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Experimentální software pro distribuci granulární syntézy do prostorového zvuku

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je vytvoření a ověření softwaru, který bude schopen pracovat s prostředky granulární syntézy a to v kontextu distribuce jednotlivých granulí nebo různě velkých mračen granulí do vícekanálového zvukového systému. Celý systém bude schopen proces kontrolovat v reálném čase nebo prostřednictvím předem definovaných funkcí. V rámci semestrální práce nástroj kompletně navrhněte a vytvořte ukázky programového řešení jednotlivých částí. V rámci navazující bakalářské práce pak realizuje plně funkční program.

DOPORUČENÁ LITERATURA:

[1] RUSS, Martin. Sound synthesis and sampling. Oxford: Focal Press, 1996. Music technology series. ISBN 0240-51429-7.

[2] The Csound book: perspectives in software synthesis, sound design, signal processing, and programming. Editor Richard BOULANGER. Cambridge: MIT Press, c2000. ISBN 0262522616.

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 26.5.2023

Vedoucí práce: MgA. Michal Indrák, Ph.D.

doc. Ing. Jiří Schimmel, Ph.D.
předseda rady studijního programu

Abstrakt

Tato práce se zabývá návrhem a realizací experimentálního softwaru, jehož úkolem je distribuce granulární syntézy do prostorového zvuku, konkrétně do oktofonního zvukového systému. Software je navržen jako VST3 zásuvný modul vytvořený pomocí frameworku JUCE. Mimo zmíněnou problematiku práce obsahuje také teoretické a technické poznatky spojené především s granulární syntézou, prostorovým zvukem a způsobem realizace výsledného softwaru. Konec této práce je věnován testovací fázi a přiblížení nedostatků výsledného softwaru.

Klíčová slova

Granulární syntéza, granule, prostorový zvuk, MIDI, JUCE, software, VST3, plugin, framework

Abstract

This thesis deals with a design and realization of the experimental software, which has the task to distribute the granular synthesis into surround sound, specifically into the octofonic sound system. The software is designed as a VST3 plugin, which was created in a JUCE framework. Apart of problematics mentioned above the thesis also contains theoretical and technical knowledge linked with granular synthesis, surround sound and a form of realization of a resulting software. The end of this thesis is dedicated to the testing phase and deficits of resulting software.

Keywords

Granular synthesis, grain, surround sound, MIDI, JUCE, software, VST3, plugin, framework

Bibliografická citace

POSPÍŠIL, Tomáš. *Experimentální software pro distribuci granulární syntézy do prostorového zvuku* [online]. Brno, 2023 [cit. 2023-05-24]. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/151144. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Michal Indrák.

Prohlášení autora o původnosti díla

Jméno a příjmení studenta:	*Tomáš Pospíšil*
VUT ID studenta:	*211807*
Typ práce:	*Bakalářská práce*
Akademický rok:	*2022/23*
Téma závěrečné práce:	*Experimentální software pro distribuci granulární syntézy do prostorového zvuku*

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: 26. května 2023 ----------

podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce MgA. Michalu Indrákovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc, za zapůjčení potřebné techniky a další cenné rady při zpracování této práce. Také děkuji za jeho časovou flexibilitu spojenou s konzultacemi a testováním výsledku této práce.

V Brně dne: 26. května 2023 ----------

podpis autora

Obsah

SEZNAM OBRÁZKŮ

Úvod

V současné době v hudebním průmyslu roste popularita vícekanálové reprodukce. Díky tomuto trendu přicházejí mnozí vývojáři VST pluginů s produkty koncipovanými pro tuto formu reprodukce. Proto bylo cílem této bakalářské práce vytvořit experimentální software pro distribuci granulární syntézy do prostorového zvuku.

V rámci semestrální práce byly navrhnuty základní principy, podle kterých byl výsledný experimentální software distribuující granulární syntézy do prostorového zvuku v této bakalářské práci realizován. Návrh experimentálního softwaru vychází z teoretických poznatků o granulární syntéze a zaměřuje se na potřeby uživatele, tedy jaké parametry by uživatel měl mít možnost nastavovat a s jakými rozhraními a technickými prostředky by mohl pracovat. Software je realizován ve formě VST3 zásuvného modulu, tedy formátu, který je podporován DAW systémy, které dokážou tento modul spustit. Součástí semestrální práce bylo také přiblížit programové řešení určitých částí dle funkčního návrhu. Realizace jednotlivých částí softwaru byla implementována ve frameworku JUCE, což je multiplatformní C++ open-source framework pro návrh a realizaci aplikací tohoto typu.

Bakalářská práce zahrnuje výsledky semestrální práce a navazuje na tuto práci s hlavním cílem doimplementovat chybějící část funkcí dle funkčního návrhu. Mimo tuto část obsahuje bakalářská práce také popis chybných přístupů zvolených při realizaci. V poslední části této práce jsou zahrnuty výsledky testování výstupního VST3 pluginu s popisem konkrétních nedostatků odhalených těmito testováními.

První kapitola obsahuje teoretické poznatky spojené s touto prací, tedy seznámení s granulární syntézou, granulemi a jejich řazením, a také s obecnou zvukovou syntézou. V další kapitole jsou pak rozebrány technické poznatky, jako je seznámení s frameworkem JUCE nebo protokolem MIDI a principem jeho komunikace pomocí MIDI zpráv. Kapitola třetí zahrnuje návrh funkčnosti VST3 pluginu z uživatelského hlediska a také návrh GUI. Kapitola čtvrtá popisuje programové řešení částí, které byly implementovány v rámci semestrální práce. V kapitole páté je podrobně rozebrána realizace zbylých funkcí, které již předmětem semestrální práce nebyly. Společně s touto realizací jsou zde popsány i přístupy, které nevedly k dokončení práce, proto bylo nutné je opustit a tuto práci pozdržely. Poslední kapitola se věnuje průběžnému i konečnému testování VST3 pluginu a také nedostatkům, které byly testováním odhaleny.

1. TEORETICKÝ ÚVOD

Tato kapitola přibližuje problematiku granulární syntézy a syntézy zvuku jako takové. Budou zde rozebrány jednotlivé metody syntézy zvuku, stavební jednotka granulární syntézy (granule) nebo princip řazení granulí, na němž je tato práce vystavěna. Dále se tato kapitola zabývá anatomií lidského ucha a principem, kterým lidské ucho dokáže lokalizovat zdroj zvuku v prostoru. Tyto poznatky slouží jako prerekvizity pro realizaci granulárního principu v prostředí JUCE a pro uvedení do prostorového zvuku.

1.1 Syntéza zvuku

Syntézu (jako pojem) můžeme chápat jako kreativní nebo tvůrčí proces, kdy se snažíme z určitých částí složit celek. Nejedná se ovšem o nahodilé skládání částí, nýbrž o promyšlené a cílené skládání částí dohromady. Syntézu zvuku tedy můžeme definovat jako proces, při kterém dochází k jeho produkci. [8]

Pro tuto produkci máme obrovské množství možností. Syntézu můžeme provádět například elektronicky nebo mechanicky, pomocí již existujících zvuků nebo můžeme použít námi definované množství oscilátorů a vytvořit výsledný zvuk složením jejich výstupů. [8]

Toto byl jen krátký výčet možností, které máme v dnešní době k dispozici. To, jakou povahu bude zvuk mít, jestli se bude jednat o komplexní zvuk nebo zvuk s jednodušší stavbou, jakou bude mít barvu, jakou bude mít povahu, to vše je pouze na preferencích operátora syntézy.

Technologický pokrok se stal doménou zejména 2. poloviny 20. století. Díky tomuto pokroku se tak otevřely dveře novým možnostem. Dřívější analogová podoba přístrojů je dnes nahrazována digitálními přístroji, které obsahují procesory a čipy a jsou schopny provádět nejrůznější úkony v minimálním časovém měřítku. V dnešní době tak můžeme jednoduše provést vlastní syntézu zvuku pomocí DAW systémů a rozšiřujících pluginů přímo z počítače, bez nutného prokázání znalostí v této oblasti.

1.2 Syntezátor

Syntezátor, jak sám název napovídá, je nástroj, pomocí kterého můžeme provádět syntézu zvuku. Pod pojmem syntezátor si mnozí vybaví obrovský analogový syntezátor, jehož moduly jsou propojeny spoustou kabelů.

Historie syntezátorů však sahá až do pravěku, kdy lidé začali používat hlas. Naše hlasové ústrojí se totiž chová jako syntezátor. Nástroje, které se po staletí formují, můžeme taktéž prohlásit za syntezátory, jelikož využívají k tvorbě zvuku čistě syntetické metody. [8]

V dnešní době bývá pod pojmem syntezátor označován elektrický hudební nástroj, tvořící zvuk syntézou a schopný produkovat široké množství zvuků. Většina syntezátorů dnešní doby disponuje širším množstvím vstupů i výstupů, které mohou být ovládány, a z nichž může být zvuk distribuován. Obsahují také určitý počet tlačítek pro ovládání syntézy v čase, některé obsahují i displeje pro výstup informací. [8]

Syntezátory můžeme dělit několika způsoby. Podle jejich vnitřního uzpůsobení je můžeme rozdělit na modulární a tzv. performance syntezátory. Dále pak můžeme syntezátory rozdělit na analogové, digitální a softwarové. [8]

1.3 Metody syntézy zvuku

Abychom mohli syntetizovat zvuk, existují různé metody, které jsou při syntéze uplatněny. Velká část těchto metod je založena na principu zdroje a modifikátoru. Tedy zdroje zvuku, který produkuje tón, který je dále modifikátorem upraven. Tím následně vzniká výsledný zvuk. Jiné metody, jako je například granulární syntéza, operují na modelech, které umí být více matematické, v případě granulární syntézy do jisté míry abstraktní. Díky míře své abstraktnosti nejsou tak známé a používané, jako například subtraktivní nebo FM syntéza, které jsou založeny na principu zdroje a modifikátoru.

Metody syntézy zvuku dělíme na metody analogové a metody digitální. Zde je dobré zmínit, že metody spojené s analogovou syntézou byly přeneseny také do roviny digitální, proto nelze tyto metody jasně přiřadit čistě k analogové nebo čistě k digitální syntéze. Při analogové syntéze jsou skládány signály, které vznikají v analogových obvodech obsahujících především oscilátory, filtry či zesilovače. V případě analogové syntézy bývá nejčastěji volen aditivní a subtraktivní přístup. Digitální syntéza pro svůj proces využívá počítače, který signály nahrazuje jejich matematickou reprezentací a s touto reprezentací dále pracuje. Výzkum v oblasti digitální syntézy stále pokračuje a hledá nové metody, jak syntetizovat zvuk. Mezi zástupce digitálních metod syntézy zvuku můžeme zařadit FM syntézu, aditivní syntézu, sample replay syntézu nebo S&S syntézu [8].

1.4 Granulární syntéza

Tato metoda je svou povahou odlišná od výše jmenovaných metod. Nedá se na ni uplatnit model zdroje zvuku a modifikátoru a její celkový princip je do jisté míry abstraktní. Mezi průkopníky této metody řadíme především Dennise Gabora a Iannise Xenakise, na které později navazují zejména Curtis Roads, Edgard Varèse nebo Barry Traux.

Při granulární syntéze je výsledný zvuk složený z krátkých segmentů, které označujeme jako granule. Tato metoda je náročná zejména pro obrovské množství parametrů, které je nutno při syntéze zohlednit. Při samotném procesu syntézy pracujeme například s délkou jednotlivých granulí, s jejich počtem v jedné periodě, jejich frekvenčním obsahem, amplitudou nebo tvarem obálky.

Curtis Roads ve své knize tvrdí: *„Jestliže n je počet parametrů pro jednu granuli a d je počet granulí v jedné sekundě, potřebujeme n krát d hodnot parametrů ke specifikování jedné sekundy zvuku.“* (Roads 2001, s.87)

Granule jsou řazeny podle určitých principů, z nichž některé budou uvedeny níže v této kapitole. Tyto principy jsou vystavěny na různých algoritmech a jsou taktéž specifikovány určitými parametry, jejichž zohlednění přidává na již zmíněné náročnosti této metody syntézy zvuku.

Granulární syntézu můžeme provádět hned několika principy, které se od sebe odlišují přístupem k řazení granulí. Mezi nejpoužívanější principy náleží synchronní a quasi-synchronní granulární syntéza, asynchronní syntéza nebo syntéza zvuku pomocí samplů. U synchronní granulární syntézy následují granule za sebou s rovnoměrným časovým intervalem, zatímco u quasi-synchronní syntézy jsou intervaly mezi granulemi různé. Tyto intervaly nejsou však úplně náhodné, doby jejich trvání se pohybují v pásmu hodnot, ze kterého tyto doby nevystoupí. Při syntéze samplů je zvuková stopa (sample) rozdělena na jednotlivé granule a přeskládána do nové podoby. Asynchronní princip, na rozdíl od synchronního a quasi-synchronního principu, není založen na skládání jednotlivých toků granulí, ale namísto toho jsou granule vrstveny do mračen a asynchronními algoritmy jsou pak jednotlivé granule vybírány a řazeny za sebe. V této práci bude řazení jednotlivých granulí prováděno na základě synchronního principu, proto mu bude věnována samostatná kapitola 1.4.3 [5].

1.4.1 Granule

Zvuková granule je nejpodstatnější částí granulární syntézy. Kombinací velkého počtu granulí jsme schopni vytvořit originální atmosférický zvuk.

Zvuková granule má dobu trvání od 1 do 100 milisekund, což z ní dělá mikroakustickou událost na prahu lidského vnímání. Každá tato granule je tvořena vlnou, která je uvnitř granule obsažena, a ADSR amplitudovou obálkou. Vlna obsažena uvnitř granule je jejím důležitým parametrem. Tyto vlny můžeme podle jejich průběhu v čase rozdělit na časově neměnné a proměnné. U časově neměnných vln se jedná především o jednoduché sinusové signály nebo o jejich součet. Naproti tomu časově proměnné vlny jsou generovány frekvenční modulací, či pomocí různých matematických technik. [5]

Úkolem amplitudové obálky granule je zamezit vzniku nechtěných rušivých elementů, jako jsou lupnutí či praskání, která vznikají z důvodu nespojitých přechodů mezi amplitudovými stavy [2]. Granule se tak svojí povahou stává věrnou zvukovou reprezentací, jelikož obsahuje informaci z časové i frekvenční oblasti [5].

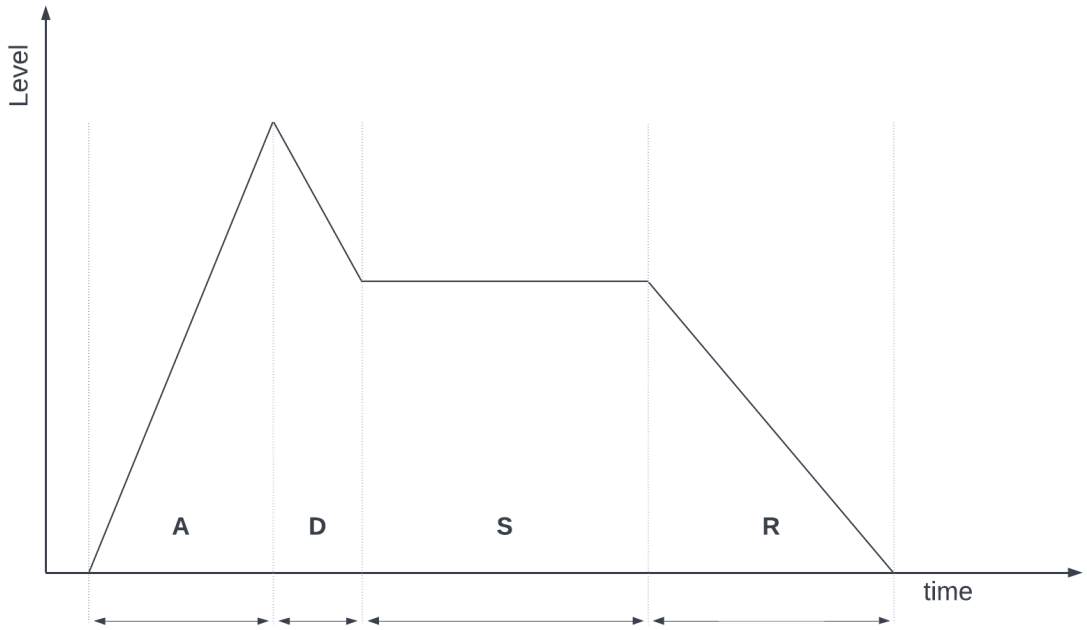
Při specifikaci granulí je potřebné myslet na čas, který lidské ucho potřebuje k zachycení zvuku. Tento čas je závislý na frekvenci. Pro vnímání frekvencí v rozmezí 1 až 2 kHz potřebuje lidské ucho přibližně 10 milisekund, pokud se dekadicky posuneme o řád níže, vzroste tento čas téměř čtyřikrát [3]. Tato skutečnost tak bývá zohledňována v délce granule a v sustainu ADSR amplitudové obálky této granule.

Z pohledu tvorby obsahu granulí existují dva způsoby, jak tohoto obsahu dosáhnout. Jedním způsobem je rozdělení audio samplu na malé, námi definované časové segmenty. Druhou variantou je pak konstrukce zvukového obsahu. Tyto metody tvorby zvukového obsahu jsou často vedeny pod názvy granulární sampling a granulární syntéza [2].

Pokud začneme granule kumulovat dohromady, nazýváme takto vzniklý soubor mračno či mrak. Tato mračna mohou teoreticky obsahovat nespočetná množství granulí, prakticky jsme však odkázáni na technické parametry výpočetního zařízení provádějícího syntézu a na paměti použité v tomto zařízení.

1.4.2 Amplitudová obálka granulí

ADSR amplitudová obálka tvaruje zvukový obsah granule v čase. Jelikož je tato obálka spojena s amplitudou, dochází při jejím průběhu k zesílení a zeslabení hlasitosti granule. ADSR je zkratkou anglických názvů pro parametry této obálky, tedy *attack, decay, sustain* a *release.*

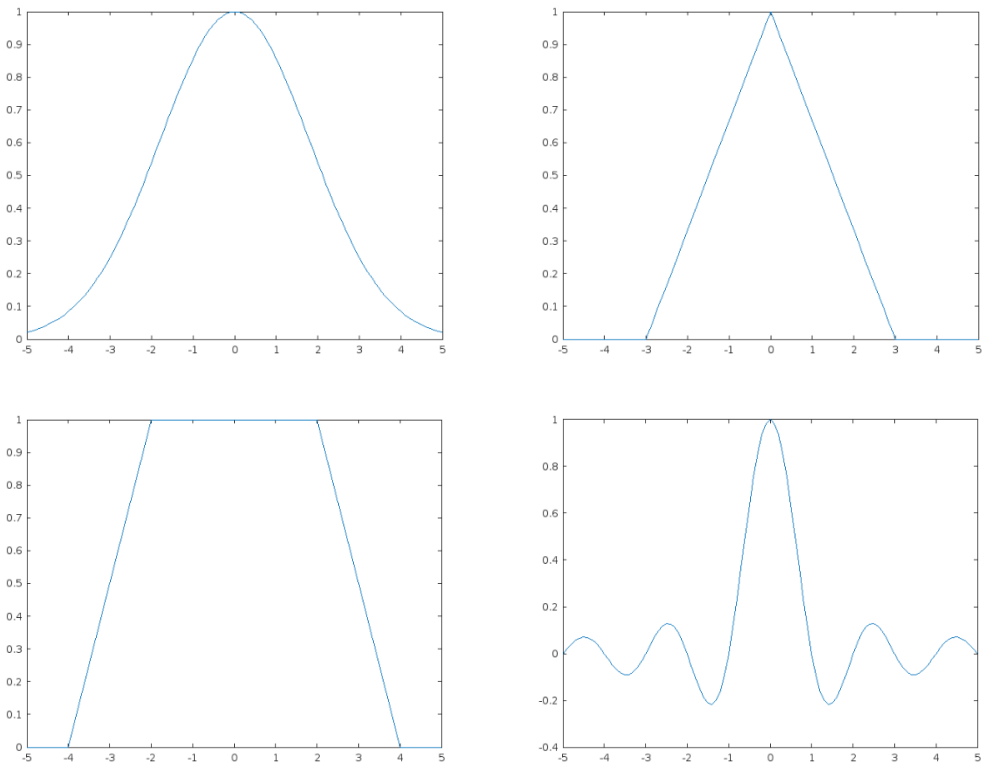


Obrázek 1.1 Nejrozšířenější podoba ADSR amplitudové obálky

- *Attack* – označuje dobu, za kterou se parametr podléhající obálce dostane z minima na své maximum
- *Decay* – jedná se o dobu, za kterou se parametr podléhající obálce dostane ze svého maxima do úrovně sustain
- *Sustain* – nese informaci o době, kde je parametr podléhající obálce neměnný
- *Release* – označuje dobu, za kterou se parametr podléhající obálce dostane z úrovně sustainu do klidového stavu

Výše je znázorněna asi nejběžnější podoba amplitudové obálky, kterou můžeme najít jako součást nejrůznějších pluginů, generátorů zvuku aj.

V granulární syntéze však našly své využití i jiné typy obálek. Jedná se o obálky, jejichž křivka se dá popsat matematickým vyjádřením. Nejčastěji se jedná o Gaussovu křivku, trojúhelníkovou nebo lichoběžníkovou křivku nebo křivku sinus cardinalis. Často používanou křivkou je také quasi-Gaussova křivka, která má stejný tvar náběhu i pádu jako Gaussova křivka, oproti ní je však obohacena o oblast sustainu. [5]



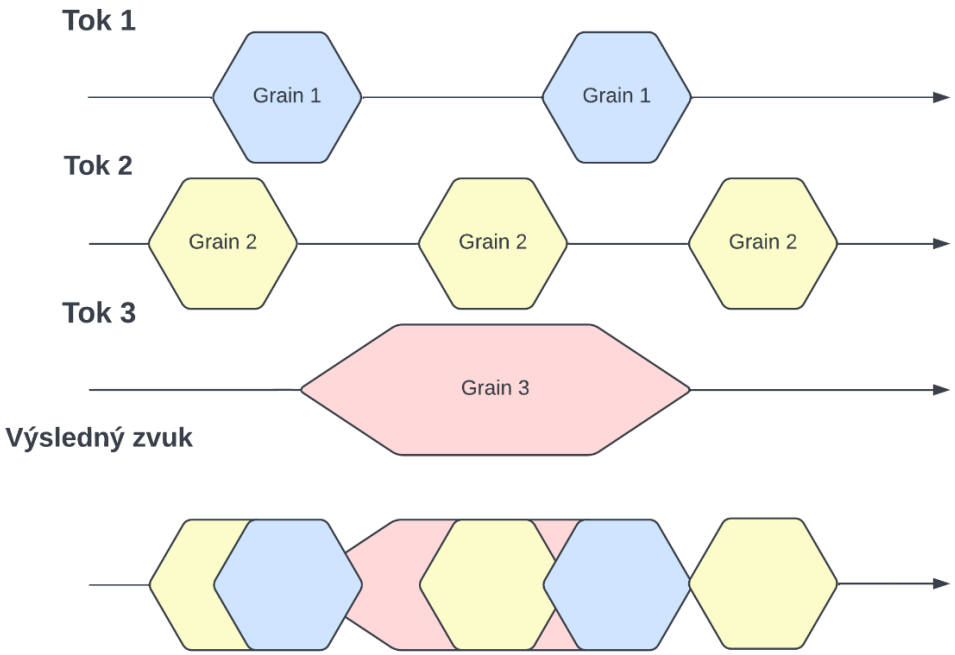
Obrázek 1.2 Amplitudová obálka tvaru Gaussovy křivky, trojúhelníkové křivky, lichoběžníkové křivky a křivky funkce sinus cardinalis

1.4.3 Synchronní granulární princip

Tento princip granulární syntézy je založen na synchronním řazení granulí. Pojmem synchronní zde rozumíme to, že jednotlivé granule na sebe navazují s lineární závislostí. Jinak řečeno jsou granule rozmístěny se specifikovaným časovým rozestupem, který se nemění, nutno podotknout že se jedná o jednu a tu samou granuli. Vzniká tak tok stejných granulí, který je v rámci syntézy zvuku směšován s jinými toky. Při tomto procesu většinou dochází k překrytí granulí, což vede k obohacení frekvenčního spektra výsledného zvuku, čímž vznikají zvuky nové. Samotný proces překrytí granulí ilustruje obrázek 1.3. [5]

Jak již bylo zmíněno, granule jsou tvarovány ADSR amplitudovou obálkou. Tvar této obálky může být dalším parametrem vstupujícím do syntézy. V dnešní době je synchronní granulární princip zprostředkováván pomocí nejrůznějších softwarových nástrojů, kde se volba tohoto parametru hojně nachází a nabízí tak uživateli možnost volby. Dalším prvkem, který je často přítomný v těchto nástrojích je nastavení doby, po jejímž uplynutí začne daný proud granulí neboli doba zpoždění daného toku granulí.

Uživatel tedy skládá různé toky granulí, které mohou mít různé amplitudové obálky a mohou mít mezi sebou různé doby zpoždění. Samotné toky mohou být taktéž zpožděné, a to nezávisle na sobě.



Obrázek 1.3 Demonstrace synchronního granulárního principu na třech nezávislých tocích granulí

1.5 Prostorový zvuk

Jak již bylo zmíněno, cílem této práce je distribuce granulární syntézy do prostorového zvuku. S ohledem na tuto skutečnost zařadím do teoretického úvodu také kapitoly popisující fungování lidského ucha a princip, na kterém lidské ucho dokáže lokalizovat zdroje zvuku.

1.5.1 Anatomie lidského ucha

Podrobná anatomie sluchového orgánu je dostupná v odborných literárních pramenech. V této podkapitole bude rozebrání sluchového orgánu zjednodušené, jelikož cílem je zaměřit se především na percepci zvuku a jeho zpracování mozkem.

Sluchový orgán můžeme rozdělit na tři základní části. Jsou to vnější ucho, střední ucho a vnitřní ucho. Vnější ucho se skládá z ušního boltce a zevního zvukovodu, který je zakončen bubínkem. Úlohou ušního boltce je směrovat zvuk do zvukovodu a zesilovat ho, konkrétně nejvíce zesílené bývají frekvence 2-3 kHz [4]. Zvukovod se svou povahou chová jako polouzavřená trubice a hromadí akustický tlak na uzavřeném konci, tedy bubínku. Tento bubínek je rozkmitáván zdrojem zvuku a přenáší tak zvuk do středního ucha.

Dominantou středního ucha jsou tři kosti, a to kladívko, kovadlinka a třmínek. Jedná se o kůstky, které vedou kmitavý pohyb dále do vnitřního ucha. Součástí středního ucha je také Eustachova trubice, sloužící k vyrovnání tlaku středního ucha s okolím tlakem.

Z vnitřního ucha by především měla být zmíněna část zvaná hlemýžď. Jedná se o část vnitřního ucha naplněnou tekutinou. Tato tekutina je v hlemýždi rozdělena Reissnerovou a bazilární membránou, je pomocí třmínku a oválného okénka rozvlňována, svým vlněním rozechvěje Cortiho orgán, obsahující vláskové buňky, které svým pohybem vysílají signál do sluchového nervu. Tento signál je pak v mozku zpracován a interpretován jako zvuk.

1.5.2 Lokalizace zdroje zvuku v prostoru

Proces lokalizace zdroje zvuku v prostoru začíná na ušním boltci. Od jeho povrchu je zvuk odrážen do vstupu zvukovodu, je zde však taky kombinován s přímou zvukovou vlnou ze zdroje. Výsledek této kombinace zvuků na vstupu zvukovodu můžeme popsat přenosovou funkcí, která má různé tvary pro různé lokace zdroje. Tato přenosová funkce je poté kombinována s fixní přenosovou funkcí zvukovodu samotného a výsledkem této kombinace se stává přenosová funkce ušního bubínku. Na tomto principu kombinace různých přenosových funkcí s fixními tak lidský mozek dokáže zasadit zvukový zdroj do prostoru. [4]

Kromě toho ušní boltec hraje svoji důležitou roli v případech, kdy se zdroje zvuku nacházejí před nebo za posluchačem. Svým tvarem se totiž chová jako zvukový filtr, kdy zvuky přicházející zpoza posluchače mají nižší vysokofrekvenční úrovně. Tímto tak mozek rozpoznává základní směr, ze kterého zvuk přichází.

Zvuk je možné slabě lokalizovat i v mediální rovině, tedy rovině vertikální procházející osou lidské hlavy. V této rovině jsou zvuky vnímány jako šířící se z určitého směru, a to nezávisle na umístění zdroje zvuku v této rovině. Frekvence kolem 300 Hz nebo 3 kHz jsou vnímány jako šířící se z pohledu před posluchačem, frekvence 1 nebo 10 kHz vnímá posluchač, jako by se zdroj zvuku nacházel za ním a frekvence 500 nebo 8000 Hz posluchač vnímá, jako by se zdroj zvuku nacházel nad jeho hlavou. [4]

Lokalizaci zvukového zdroje v prostoru napomáhá také binaurální slyšení, kdy dochází k lokalizaci zdroje zvuku v horizontální rovině. Při šíření zvukové vlny klesá její intenzita se vzdáleností, nemluvíme-li o rovinném vlnění, které má ve všech bodech roviny stejnou intenzitu. Díky vzdálenosti uší od sebe tak dochází k poklesu intenzity na vzdálenějším uchu oproti uchu, které se nachází blíže ke zdroji. Dalším faktorem je také časové zpoždění, které je dáno vzdáleností mezi ušima. Jistou úlohu zde sehrává i akustický stín hlavy.

Na frekvencích pod 1 kHz vnímá lidský mozek zejména časovou diferenci, nad touto frekvencí pak lokalizujeme zdroj zejména pomocí intenzitní diference. [4]

Člověk by však čistě na základě tohoto principu nebyl schopen rozlišit lokalizaci zdrojů zvuku nacházejících se přímo před ním a přímo za ním. V kombinaci s již zmíněnými principy se však jedná o důležitou vlastnost, která výrazně napomáhá lokalizaci zdroje zvuku v prostoru.

2. TECHNICKÝ ÚVOD

Tato kapitola se zabývá technickou částí této práce. Kapitola zahrnuje seznámení s komunikačním protokolem MIDI, principem jeho komunikace pomocí zpráv a také zde budou rozebrány druhy zpráv, pomocí kterých výsledný software komunikuje. Jedná se o zprávy *Nota zapnuta* a *Nota vypnuta*.

Výsledný experimentální software je koncipován jako VST3 zásuvný modul, a proto bude i tento formát přiblížen společně s frameworkem JUCE (aplikací *Projucer*), který svou koncepcí umožňuje vytvářet tyto rozšiřující zásuvné moduly, a to především pro DAW systémy.

Posledním tématem této kapitoly pak bude seznámení s prostorovým zvukem, konkrétně s oktofonním zvukovým systémem, jelikož výsledný software pracuje s rozmístěním zvukového materiálu do osmi kanálů v prostoru.

2.1 Protokol MIDI

Protokol MIDI (*Musical Instrument Digital Interface*) je transportní protokol, který umožňuje přenos informací mezi audio zařízeními jako jsou kontroléry, klávesy nebo počítače, a to v reálném čase. Tento protokol komunikuje na základě zasílání zpráv a pro zajištění správné komunikace mezi různými zařízeními byl tento protokol standardizován.

Tento protokol je přenášen po MIDI rozhraní, které je tvořeno 5-pinovými DIN konektory. Přenos protokolu po těchto konektorech probíhá pouze v jednom směru, proto jsou zařízení vybavena vstupy IN a OUT, některá obsahují i konektor THRU, na který jsou přenášena data ze vstupu a pomocí něhož mohou být data přenesena do dalšího zařízení. V současnosti se však využívá spíše rozhraní USB, konkrétně se jedná o kabel zakončený na jedné straně USB typu A a na druhé straně USB typu B. Toto rozhraní umožňuje obousměrnou komunikaci. [11]

Jednotlivé MIDI zprávy mají svoji standardizovanou strukturu. Každá zpráva je binárního charakteru a je složena ze stavové 8-bitové informace (*Status byte*) a několika datových 8-bitových informací (*Data byte*). To, zda se jedná o stavový nebo datový byte, je závislé na nejvíce významném bitu (*MSB*). Pokud má tento bit hodnotu 1, jedná se o stavový byte, pokud má hodnotu 0, jedná se o byte datový.

U stavového bytu následuje po *MSB* 3-bitová informace o typu kanálových dat, za ní pak 4-bitová informace o kanálu, na kterém bude zařízení přijímající informaci „naslouchat“ těmto zprávám. [11]

Datový byte je tvořen *MSB* s hodnotou 0, za kterým následuje 7-bitová informace. Základní kanálová data mají buď jeden, nebo dva datové byty. Po výsledném softwaru je požadována reakce na zprávy Nota zapnuta a Nota vypnuta, které mají obě 2 datové byty. Tyto MIDI zprávy budou rozebrány v podkapitolách níže.

2.1.1 MIDI zprávy Nota vypnuta a Nota zapnuta

Obě tyto zprávy spadají do kategorie kanálových zpráv. Konkrétně se jedná o hlasová data, což znamená, že stavový byte obsahuje informace o kanálu, na kterém jsou tyto zprávy přenášeny. Pro obě tyto zprávy platí, že za stavovým bytem se nacházejí dva datové byty, na kterých jsou zaznamenány důležité informace pro danou zprávu

Zpráva *Nota zapnuta* má binární identifikátor 001, který se nachází za nejvýznamnějším bitem statusového bytu a je následován binárním vyjádřením čísla kanálu. Za stavovým bytem následuje první datový byte, který přenáší informaci o výšce noty, která byla zmáčknuta na kontroléru či klávesách. Této informaci je vyhrazeno 7 bitů. Noty jsou tedy zakódovány pomocí 128 hodnot, přičemž notě C jednočárkované oktávy náleží hodnota 60. Druhý datový byte obsahuje rychlostní data, která se pojí s dynamikou a jsou taktéž zakódována na 7 bitech. Obecně platí, že čím větší aritmetická hodnota bude na těchto 7 bitech uložena, tím větší bude dynamika výsledné noty, viz obrázek 2.1. Celková struktura MIDI zprávy *Nota zapnuta* je znázorněna na obrázku 2.2, kde každé X označuje binární hodnotu 0 nebo 1. [10] [11]

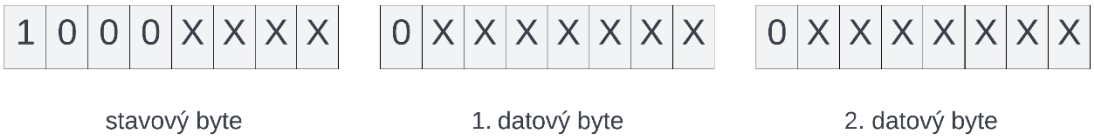
Obrázek 2.1 Grafické znázornění principu převodu rychlostních dat na dynamiku ve 2. datovém bytu zprávy Nota zapnuta

Obrázek 2.2 Grafické znázornění struktury MIDI zprávy *Nota zapnuta*

Zpráva *Nota vypnuta* má binární identifikátor 000, kromě kterého se ve stavovém bytu nachází opět informace o kanále, po kterém je komunikováno. Za stavovým bytem se nachází první datový byte označující výšku noty a druhý datový byte, který nese informaci o rychlosti, s jakou byla klávesa uvolněna. [11]

Princip komunikace pomocí těchto zpráv je poměrně zřetelný. Pokud bude zmáčknuta klávesa na kontroléru či klávesách, bude zaslána zpráva *Nota zapnuta*. Tato zpráva nese informaci o tom, jaká klávesa a s jakou dynamikou byla zmáčknuta. Nota bude trvat tak dlouho, dokud nedojde k uvolnění klávesy, čímž dojde k zaslání zprávy *Nota vypnuta* pro příslušnou notu a nota bude ukončena s ohledem na rychlost uvolnění této klávesy.

Na obrázku 2.3 je znázorněna struktura zprávy *Nota vypnuta*.



Obrázek 2.3 Grafické znázornění struktury MIDI zprávy *Nota vypnuta*

2.2 VST3 standard

VST3 je nejaktuálnější a momentálně nejrozšířenější verzí standardu VST (*Virtual Studio Technology*), se kterým přišla na trh firma Steinberg Media Technologies v roce 1996. Toto rozhraní bylo firmou Steinberg vyvinuto s cílem umožnit uživateli importovat do DAW systémů virtuální nástroje a efekty, díky kterým bude moci počítač kompletně zastoupit funkci nahrávacích studií a umožní uživateli produkci bez nutnosti použití fyzických zařízení. Jedná se o otevřený standard, který umožňuje firmám i jednotlivcům vyvíjet rozšíření pro DAW systémy v tomto formátu.

VST plugin přijímá od hostitelské aplikace (DAW) tok zvukových dat, která jsou posílána v blocích určité velikosti. Dále VST plugin zvuková data zpracovává a po zpracování procesorem předává výstupní zvuková data zpět hostitelské aplikaci. Hostitelská aplikace vnímá VST plugin jako „černou skříňku“, která má libovolný počet vstupů a výstupů, a o které nepotřebuje znát žádné informace pro její použití.

Formát VST3 vychází z předešlých VST1 a VST2, se kterými není zpětně kompatibilní, ovšem přebírá a zdokonaluje určité jejich části a přidává nové možnosti. Mezi tyto možnosti náleží například podpora zvětšení obsahu GUI pluginu, odeslání informace o tom, jaký parametr se zrovna nachází pod myší, podporu ovládání funkcí pluginů pomocí dálkových ovladačů a mnoho dalších.

Firma Steinberg Media Technologies uvedla formát VST3 v roce 2008, v současné době stále probíhá jeho vývoj a odstraňování nedostatků. Pro realizaci projektů v tomto formátu je k dispozici VST3 SDK (*Virtual Studio Technology Software Development Kit*) v aktuální verzi 3.7.8. [12]

2.3 Framework JUCE

JUCE je open-source C++ framework, který umožňuje vývoj aplikaci jak pro počítače, tak pro mobilní zařízení. Tento framework má velkou oblibu zejména mezi tvůrci audio pluginů, jelikož svojí koncepcí umožňuje tvorbu ve formátech VST3, AU, AUv3, AAX, LV2 nebo VST. Podporuje také přenositelnost mezi platformami, tedy mezi platformou Windows, Linux či macOS. Pro začátečníky bylo na oficiálních webových stránkách vývojáři připraveno přes 60 edukativních projektů[1], mimo které webové stránky obsahují i odkaz na oficiální fórum[2] s aktivní komunitou.

Framework je obsluhován skrze aplikaci *Projucer*. Jedná se o jednoduchou aplikaci, jejímž úkolem je vygenerovat projekt podle nastavení uživatele a připravit tento projekt pro spuštění v programovacích prostředích jako jsou *Visual Studio*, *XCode* nebo *Code::Blocks*. Projekt je generován pomocí šablon, ze kterých uživatel volí, jaký typ aplikace, pluginu či knihovny chce vytvořit.

[1] https://juce.com/learn/tutorials
[2] https://forum.juce.com/

Každá tato šablona obsahuje specifické zásuvné moduly, které jsou k ní ve výchozím stavu připojeny. Uživatel má možnost v případě potřeby tyto moduly rozšířit o jím vybrané. Šablony mají svá odlišná nastavení, která se odvíjí od jejich funkce. U šablony typu plugin, která je pro tuto práci použita, může uživatel dále specifikovat například výstupní formát (*VST3, AU, AXX*) či zahrnout MIDI vstupy a výstupy, případně upřesnit jejich počet. Po této specifikaci je pak projekt generován a otevřen v preferovaném programovacím prostředí.

Jelikož se jedná o C++ framework, vychází programovací princip z objektově orientovaného modelu. Jednotlivé objekty (třídy) obsahují své vlastní proměnné a metody, pomocí kterých lze s objekty komunikovat a jsou popsány v digitální dokumentaci[3] na oficiálních webových stránkách[4].

2.4 Oktofonní zvuk

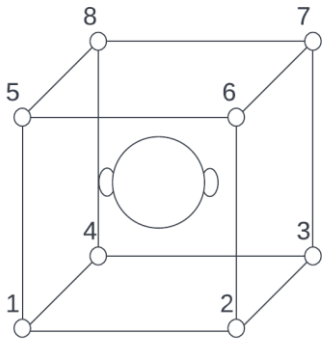
Oktofonní zvuk je název pro formu vícekanálové prostorové zvukové reprodukce, při které je zvukový signál distribuován do osmi diskrétních kanálů, jejichž jednotlivé výstupy jsou vedeny do osmi zvukových reproduktorů. Cílem této formy je umožnit lépe rozprostřít zvuky do prostoru a dodat posluchači hlubší prožitek z reprodukovaného materiálu. [6]

Tato forma je počtem kanálů dosti podobná standardu prostorového zvuku 7.1, u které je ovšem jeden z kanálů určen především pro přenos nízkých frekvencí. Dále se tento standard odlišuje také rozestavením jednotlivých kanálů v prostoru. [9]

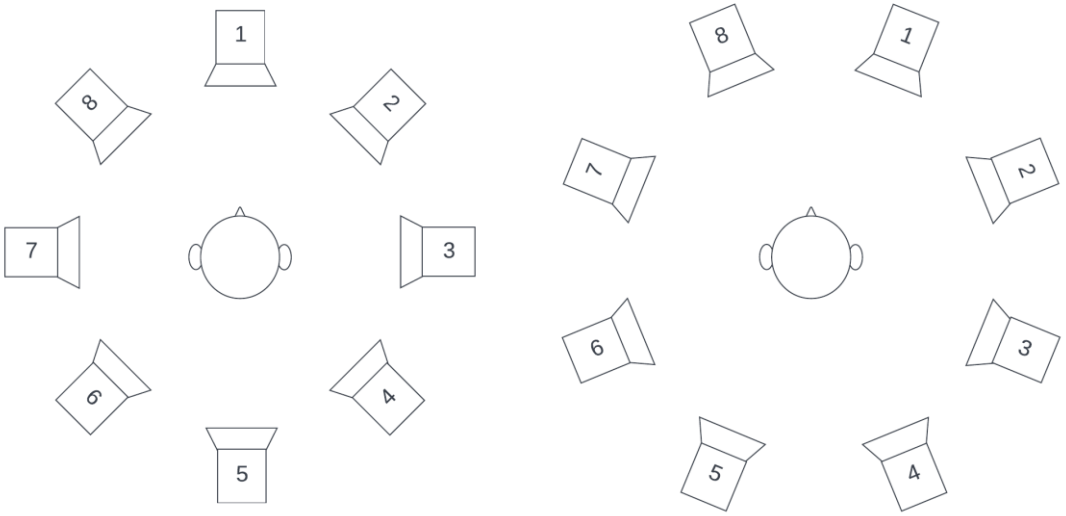
Jednotlivé reproduktory mohou být kolem posluchače rozmístěny ve tvaru hranolu, kde se na každém rohu tohoto hranolu nachází jeden z reproduktorů směřující do středu tohoto tělesa, ve kterém se posluchač nachází. Jedná se o model, který vychází z kvadrofonní formy, kde je zvuk distribuován do čtyř reproduktorů rozmístěných do tvaru čtverce. Oproti této formě je však doplněn o třetí dimenzi. Jedná se tedy o dva kvadrofonní systémy nad sebou, čímž je do zvuku vnášen pocit výšky. Rozložení reproduktorů je znázorněno na obrázku 2.4. O vývoj a realizaci této formy se zasloužily zejména Hubert Caron a Daniel Laberge. [7]

Dalším typem rozmístění reproduktorů je kruhové, nebo spíše osmiúhelníkové rozmístění, při kterém opět reproduktory směřují do středu tohoto útvaru, kde se nachází posluchač. Reproduktory jsou nejčastěji rozmístěny tak, že jsou od sebe vzdáleny v úhlu 45° s prvním reproduktorem umístěným proti posluchači. Dalším způsobem je pak rozmístění reproduktorů, kdy první reproduktor je před posluchačem pod úhlem 22,5°, a zbylé reproduktory jsou rozmístěny opět o 45° rovnoměrně od sebe. Jednotlivá rozestavení reproduktorů jsou uvedena na obrázku 2.5. [6]

[3] https://docs.juce.com/master/index.html
[4] https://juce.com/



Obrázek 2.4 Grafické znázornění rozestavení reproduktorů pro reprodukci oktofonního zvuku vycházející z kvadrofonního rozestavení



Obrázek 2.5 Grafické znázornění kruhového nebo též osmiúhelníkového rozestavení reproduktorů při reprodukci oktofonního zvuku.

3. NÁVRH GUI A FUNKČNOSTI VST PLUGINU

V této kapitole je popsán návrh výsledného zásuvného modulu, který zahrnuje navržení funkčnosti z uživatelského hlediska a také návrh grafického uživatelského rozhraní.

V kapitole 3.1 je popsáno, jakým způsobem by měl software pracovat, jakým způsobem by ho měl uživatel ovládat a co by od tohoto softwaru měl očekávat. Bude zde podrobně rozebráno, které parametry bude mít uživatel možnost volit a s jakými omezeními. Dále kapitola obsahuje návrhy toku zvuku, tedy 2D vzory, jejichž název evokuje povahu toku zvuku. Jedná se o algoritmy, které určují pořadí spuštění jednotlivých kanálů reprodukčního systému.

Kapitola 3.2 se zaměřuje na návrh grafického uživatelského rozhraní, tedy na to, jakým způsobem budou jednotlivé prvky rozmístěny uvnitř okna pluginu. Pro tuto část byla zhotovena předběžná vizualizace v prostředí *Projucer*, která zachycuje orientační rozmístění prvků v rozhraní.

3.1 Návrh funkčnosti zásuvného modulu

Jak již bylo výše v této práci avizováno, cílem experimentálního softwaru je distribuce granulární syntézy do prostorového zvuku. Distribuce bude probíhat do osmi diskrétních kanálů. Experimentálnost tohoto softwaru je založena na nastavování parametrů granulární syntézy a jejich promítání do jednotlivých kanálů. Na každém kanálu tak vznikne uskupení uživatelem definovaných granulí, které budou s ohledem na jednotlivé parametry mixovány a přehrávány. Algoritmus, kterým budou tyto granule přehrávány, bude taktéž volen uživatelem z předem definované nabídky vzorů. Na základě těchto vzorů bude výstup zvuku z jednotlivých reproduktorů v daných okamžicích vizualizován pomocí změny barvy komponentů reprezentujících tyto reproduktory. Další funkcí bude možnost odstranění granule z určitého kanálu.

3.1.1 Nahrání a přehrání zvukového materiálu jako zdroje granulární syntézy

Zdrojem granulí budou zvukové soubory. Aby bylo možné zvukové sobory vybrat a použít, je nutné, aby měl uživatel možnost otevřít knihovnu s nabídkou souborů podporovaných formátů. GUI tedy bude obsahovat tlačítko, po jehož stisknutí se vyvolá dialogové okno knihovny souborů. Podporovanými formáty budou zvukové soubory s příponami .wav a .aiff. Po zvolení souboru se vykreslí zmenšený zvukový průběh reprezentující obsah daného souboru.

Společně s tímto průběhem se vykreslí také kurzor, který bude sloužit pro nastavení pozice přehrávání, které bude možné spustit, zastavit a vrátit do počáteční pozice pomocí dvou tlačítek. Pozice kurzoru bude stejně jako tlačítka ovládána stisknutím levého tlačítka myši.

Dalším způsobem nahrání souboru bude možnost zvaná „Drag and Drop“. Jedná se o funkci, kdy má uživatel možnost vybrat soubor z otevřené knihovny souborů a při stisknutém tlačítku myši může přesunout soubor do okna pluginu. Zde si bude muset uživatel sám ohlídat, jakého formátu je daný soubor. Jak již bylo řečeno, výsledný plugin bude podporovat pouze formáty .wav a .aiff. Soubory jiného formátu tak nebudou načteny a bude k nim přistupováno, jako by uživatel žádný soubor nenahrál.

3.1.2 Parametry granulí, nastavení a tvorba granule

Po nahrání souboru bude od uživatele očekáváno vytvoření granule. Granule je koncipována jako objekt, uvnitř kterého budou uchovány všechny důležité informace. Každá granule ponese informaci o amplitudové obálce, která na ni bude aplikována. Cílem je implementovat tři amplitudové obálky, tedy obálku tvaru Gaussovy křivky, obálku tvaru lichoběžníku a obálku tvaru trojúhelníku. S ohledem na časové dispozice při realizaci práce může být později počet obálek rozšířen. Amplitudová obálka bude volena uživatelem v GUI pomocí posuvníku, kde dané polohy budou odpovídat jednotlivým obálkám.

Další informací, kterou bude tento objekt obsahovat, je informace o délce trvání granule v milisekundách. Tato délka bude rovněž volena posuvníkem s krokem jedna milisekunda, pod kterým uživatel najde nastavenou hodnotu. Počátek granule je stanoven kurzorem, viz kapitola 3.1.1. Po nastavení doby trvání granule bude vymezen ve zvukovém průběhu úsek odpovídající této granuli.

Další parametr ovládaný posuvníkem bude vymezovat, kolik granulí zazní v jedné sekundě při reprodukci. Hodnota tohoto parametru bude závislá na délce granule, tedy pokud zvolí uživatel delší granule, bude moci zvolit méně granulí v jedné vteřině a naopak.

Posledním parametrem je časové odsazení granule od počátku procesu. Uživatel bude moci nastavit čas, po jehož uplynutí granule poprvé vstoupí do syntézy. Parametr časového offsetu bude pro tuto práci omezen na pět minut, nastavení bude možné s krokem jedna sekunda.

Pod posuvníky výše deklarovaných parametrů se bude nacházet řada přepínačů. Každý z těchto přepínačů bude sloužit jako identifikátor příslušného kanálu, na který bude granule přidána. Po zvolení příslušného kanálu bude muset uživatel potvrdit svůj výběr potvrzovacím tlačítkem, čímž se zaznamenají všechny nastavené parametry do objektu granule a adresa objektu v paměti bude přidána na preferovaný kanál.

Poslední možností uživatele v této části bude možnost obnovit původní stav, všechny parametry tak budou nastaveny do svých výchozích pozic.

3.1.3 Odstranění granule

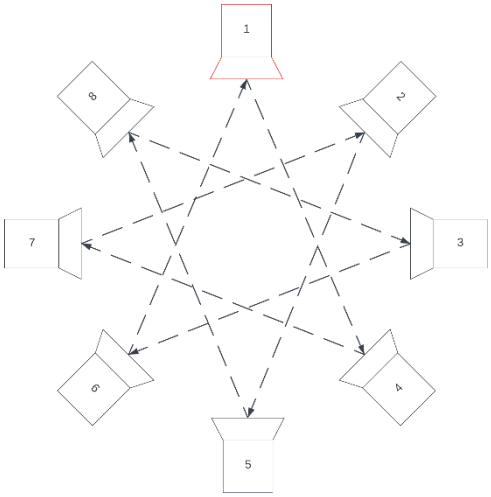
Uživatel bude mít možnost granule z daného kanálu odstranit. Pro každý kanál bude vytvořen rozbalovací seznam, jehož položkami budou jednotlivé granule. Při potřebě odstranit granuli z daného kanálu uživatel vybere granuli z rozbalovacího seznamu a stiskne potvrzovací tlačítko, po jehož zmáčknutí budou granule vymazány. Toto bude možné provést pro všechny kanály současně, současně tedy bude možné odstranit osm granulí z osmi kanálů. Odstranění dvou granulí z jednoho kanálu ve stejný okamžik nebude možné, vždy bude možné odstraňovat jen jednu granuli v daný okamžik.

Tato implementace bude mít jistou nevýhodu. Jednotlivé granule budou v rozbalovacích seznamech řazeny podle toho, jak byly na kanál přiřazeny. Uživatel tedy bude muset mít přehled o tom, která z granulí je tou, kterou chce odstranit.

3.1.4 Návrhy toku zvuku

Při reprodukci zvuku bude zvuk vyzařovat z daných kanálů na základě určitých algoritmů. Pokud bychom se na šíření zvuku z reproduktorů podívali ve 2D pohledu, mohli bychom tento tok zvuku přirovnat k určitým tvarům. Jednotlivé algoritmy jsou pojmenovány podle těchto tvarů, jsou to algoritmy *Hvězda*, *Půlkruh*, *Přesýpací hodiny* a *Kruh*. Níže jsou jednotlivé algoritmy rozebrány a ilustrovány.

Algoritmus *Hvězda* bude nejkomplexnějším algoritmem, ve kterém budou zapojeny všechny použité kanály a jehož výsledkem může být mohutný zvuk. Reprodukce započne před uživatelem na 1. kanálu, další dva budou vynechány a reprodukce bude pokračovat na 4. kanálu. Poté znovu dojde k vynechání dvou kanálů a reprodukce bude pokračovat na 7. kanálu. Tímto způsobem se bude pokračovat, dokud nebude opsán tvar hvězdy.

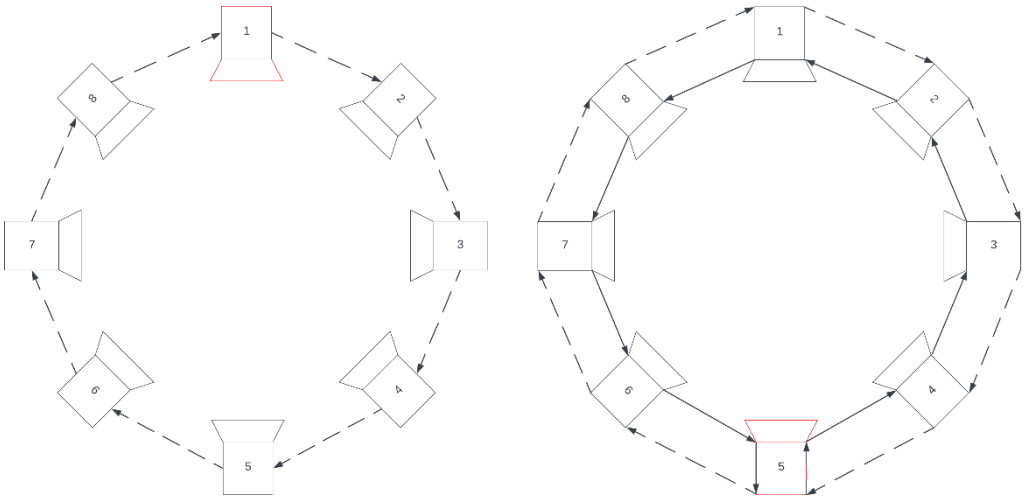


Obrázek 3.1 Ilustrace principu algoritmu Hvězda

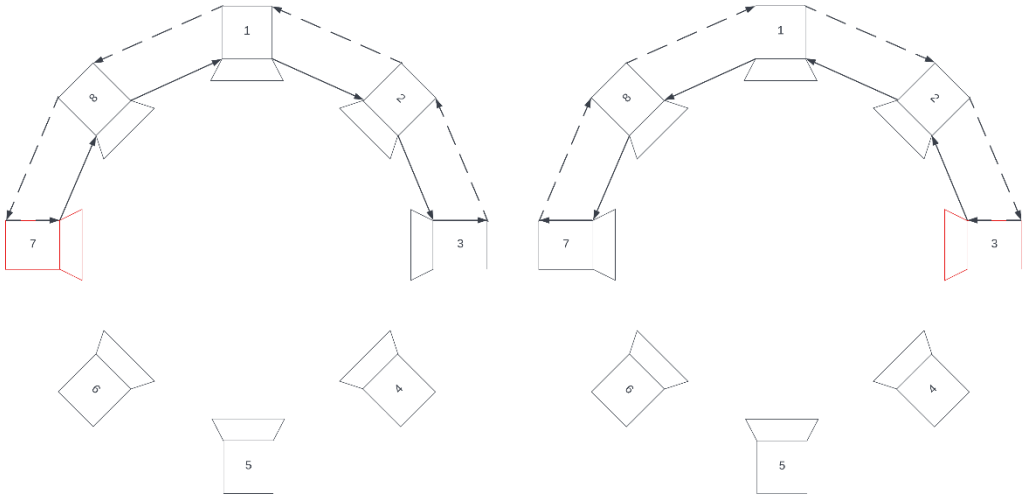
Dalším algoritmem bude algoritmus *Kruh*, respektive se bude jednat o dva kruhové algoritmy – *Kruh* a *Kruh (přelévavý)*. Pro algoritmus *Kruh* bude zvuk putovat v kruhu od 1. kanálu k 8. pořád dokola. Výsledný zvuk bude mít za cíl navodit pocit rotace. Pro algoritmus *Kruh (přelévavý)* započne reprodukce na 5. kanále umístěném za uživatelem, zvuk bude putovat z pravé strany před uživatele a následně z levé strany zpět za něj, kde se tok zvuku obrátí a zvuk se stejnou cestou vrátí zpět za uživatele. Oba tyto algoritmy jsou ilustrovány na obrázku 3.2.

Na obdobném toku zvuku, který se přelévá z jedné strany na druhou, je vystavěn i algoritmus *Půlkruh*, při kterém je reprodukce přesunuta před posluchače. Reprodukce započne buď na pravém (*Půlkruh (zprava)*) nebo na levém uchu posluchače (*Půlkruh (zleva)*) a bude pokračovat z dané strany před posluchačem na stranu protější. Princip algoritmu je zachycen na obrázku 3.3.

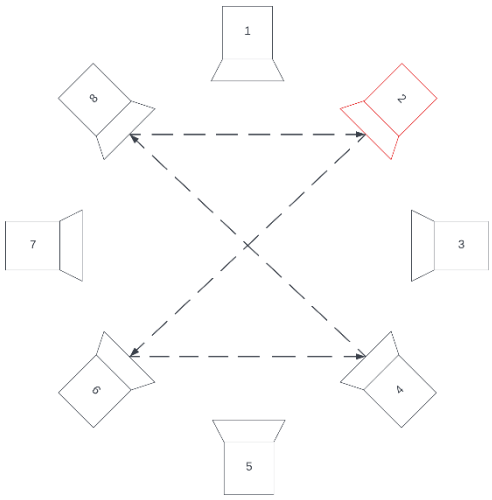
Posledním algoritmem je algoritmus nazvaný *Přesýpací hodiny* podle tvaru, který 2D reprezentace toku zvuku připomíná. Reprodukce začne na 2. kanálu vpravo před uživatelem, bude pokračovat na 6. kanál vlevo za uživatelem, poté na 4. kanál opět za uživatelem, nyní však vpravo, a nakonec se zvuk přemístí opět před uživatele na 8.kanál vlevo. Zvuk tedy bude navozovat pocit přelévání z popředí do pozadí uživatele. Tento algoritmus přibližuje obrázek 3.4.



Obrázek 3.2 Ilustrace algoritmu *Kruh* (vlevo) a *Kruh přelévavý* (vpravo)



Obrázek 3.3 Ilustrace algoritmu *Půlkruh(zleva)* (vlevo) a *Půlkruh(zprava)* (vpravo)



Obrázek 3.4 Ilustrace principu algoritmu *Přesýpací hodiny*

3.1.5 Vizualizace toku zvuku

Tato funkce napomůže uživateli vizuálně rozlišit, ze kterého z osmi kanálů v daném okamžiku vychází zvuk. Každý kanál bude reprezentován pro jednoduchost tlačítkem, které bude mít potlačenou možnost stisku. Pokud začne z daného kanálu vycházet zvuk, změní tlačítko své pozadí na jinou barvu, než je jeho výchozí stav. Tímto vznikne vizuální reprezentace výše navržených algoritmů.

3.1.6 Ovládání reprodukce protokolem MIDI

Spuštění a přerušení reprodukce bude uživatel schopen kontrolovat pomocí kontroléru, kláves či jiných zařízení komunikujících pomocí MIDI protokolu, respektive odesílajících MIDI zprávy *Nota zapnuta* a *Nota vypnuta*. Pro tuto funkci bude v prostředí *Projucer* při generaci povolen MIDI vstup i výstup. GUI bude obsahovat rozbalovací seznam pro výběr kompatibilního zařízení pro komunikaci skrze MIDI protokol.

V případě, že uživatel nebude vybaven tímto typem zařízení, nebo by připojené zařízení chybělo v seznamu kompatibilních zařízení, bude GUI obsahovat miniaturu kláves, které se budou chovat jako fyzické zařízení.

Koncept celé reprodukce stojí na výše zmíněných MIDI zprávách. Při stisku libovolné klávesy bude odeslána zpráva *Nota* zapnuta, čímž se spustí reprodukce v závislosti na zvoleném algoritmu, viz kapitola 3.1.4. Pokud uživatel uvolní klávesu, bude zaslána zpráva *Nota vypnuta*, čímž dojde k přerušení reprodukce.

3.2 Návrh rozložení GUI

Grafické uživatelské rozhraní bude rozděleno vertikálně na dvě části. Levá část okna bude obsahovat prvky spojené s načtením zvukové stopy, grafickou reprezentaci jejího obsahu, jednotlivé posuvníky pro nastavení parametrů granulí a prvky pro odstranění granulí. Dále také bude obsahovat výběrový seznam pro volbu MIDI vstupního zařízení.

Pravá část okna GUI bude obsahovat tlačítka pro volbu algoritmů, na jejichž základě bude reprodukce probíhat. Pod těmito tlačítky bude rozmístěno osm tlačítek reprezentujících jednotlivé kanály. Tato tlačítka budou změnou svého pozadí vizualizovat tok zvuku těmito kanály. Pro lepší pochopení algoritmů bude u každého možnost přehrát jeden cyklus průchodu zvuku.

Pod oběma částmi se bude v celé šířce okna nacházet klaviatura schopná nahradit fyzické zařízení pro přenos MIDI zpráv.

Všechny výše jmenované položky budou uzavřeny ve vlastních blocích nazvaných výstižně podle funkce daného bloku. Dále se bude pod všemi posuvníky zobrazovat nastavovaná hodnota, tedy například pro délku granule počet milisekund jejího trvání. GUI by mělo obsahovat také informační tlačítko, po jehož zmáčknutí by se mělo zobrazit informační okno s informacemi k softwaru a jeho použití.

Na obrázku 3.5 se nachází vizualizace tohoto návrhu GUI. Návrh byl vypracován pomocí jednotlivých tříd frameworku JUCE a všechny komponenty, které jsou na obrázku zachyceny, mají pouze informační charakter, stejně jako hodnoty jednotlivých parametrů. GUI je navrženo v angličtině, jelikož se jedná o jeden z mezinárodních jazyků.

Obrázek 3.5 Návrh grafického uživatelského rozhraní výsledného softwaru

4. PROGRAMOVÉ ŘEŠENÍ VYBRANÝCH ČÁSTÍ

V této kapitole je rozebrán princip funkčnosti a objektů jednotlivých částí kódů, které byly naprogramovány v rámci semestrální práce. Jedná se o samostatné projekty vycházející z oficiálních tutoriálů uvedených na stránkách frameworku JUCE. Tyto samostatné projekty byly později seskládány do jednoho celku a použity dále v této práci, v téměř totožné podobě. Semestrální práce se zaměřovala zejména na realizaci základních funkcí z pohledu uživatele, tedy na možnost nahrání zvukového materiálu pomocí tlačítka či pomocí operace *„Drag and drop"*. Dále se semestrální práce zabývala problematikou vytvoření bufferů pro nahraná zvuková data a zkoumala pravidla pro kopírování obsahu mezi nimi. Poslední projekt z této sady přibližuje komunikaci se zásuvným modulem pomocí protokolu MIDI. V následujících podkapitolách budou jednotlivé části podrobněji rozebrány s odkazy na příslušné přílohy.

4.1 Nahrání zvukového materiálu a jeho přehrání

Aby bylo možné pomocí zásuvného modulu provést granulární syntézu, je podle funkčního návrhu nutné, aby uživatel nahrál zvukovou stopu v podobě zvukového souboru. To je umožněno použitím třídy *FileChooser*, která umožňuje metodou *launchAsync()* vyvolat dialogové okno pro výběr zvukového souboru. Po vyvolání tohoto okna je nutné ohlídat, aby se jednalo o zvukové soubory, což nám zajišťuje objekt *AudioFormatManager*, který metodou *registerBasicFormats()*zavede formáty .wav a .aiff jako preferované. [17] [26]

Dialogové okno tak zobrazí pouze soubory s těmito příponami. Pro přečtení tohoto souboru je vytvořen objekt *AudioFormatReader*, pomocí kterého objekt *AudioFormatReaderSource* načte obsah zvukového souboru společně s informací o vzorkovací rychlosti daného souboru. [18] [19]

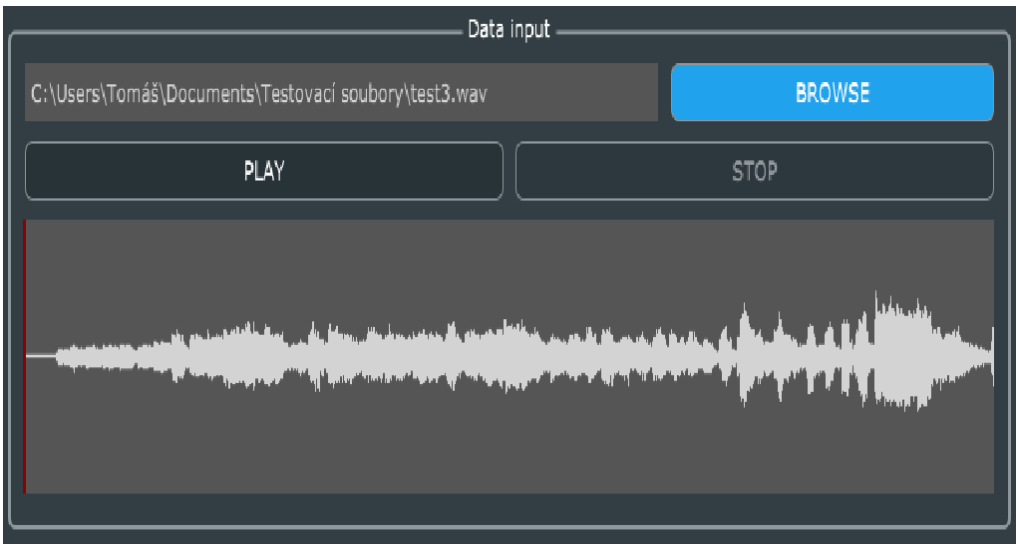
Hlavní úlohu celého procesu nahrávání a přehrávání zvukového souboru má objekt typu *AudioTransportSource*, který umožňuje, aby byl obsah předaný konstruktoru tohoto objektu přehrán, pozastaven nebo umožňuje například nastavení pozice přehrávání. Obsah čítače *AudioFormatReader* je předán objektu typu *AudioTransportSource*, objektu jsou tak přidána zvuková data. [18] [23]

Dále program hlídá změnu stavu pomocí enumerátorů, které nesou informaci o stavu, ve kterém se zásuvný modul nachází, tedy jestli je nastaveno započetí přehrávání obsahu, jestli se obsah již přehrává, jestli dochází k jeho pozastavení nebo jestli už bylo přehrávání úplně zastaveno.

Pro lepší orientaci uživatele v přehrávaném materiálu obsahuje program objekt *AudioThumbnail*, jehož úlohou je vykreslit obsah zvukového souboru, který mu byl při procesu čtení předán. Vykreslení je provedeno v podobě zvukového průběhu ve zmenšeném měřítku. [22]

Pro lepší pochopení jsou na konci této práce uvedeny přílohy, ve kterých je uveden zdrojový kód vystihující obsah této podkapitoly. Příloha A.1 demonstruje funkci načítající zvukový soubor, příloha A.2 přibližuje nastavení dialogového okna a volání funkce z přílohy A.1. V příloze A.3 je uveden kód, který reaguje na změněný enumerátor.

Obrázek 4.1 zachycuje uživatelem vybraný zvukový soubor, jehož obsah byl zachycen pomocí zvukového průběhu a připraven pro přehrávání.



Obrázek 4.1 Nahrání a vizualizace zvukového souboru pro další možnou činnost programu

4.2 Realizace funkce „Drag and drop“

Aby bylo možné vyhnout se zdlouhavému procesu, kdy musí uživatel zmáčknout tlačítko pro otevření dialogového okna a vybrání konkrétního souboru, umožňuje software funkci výběru souboru z otevřené knihovny a se stlačeným levým tlačítkem myši jeho přesunutí do okna jedoucího zásuvného modulu. Pro zajištění této funkce obsahuje framework JUCE třídu *FileDragAndDropTarget*, která obsahuje metody *isInterestedInFileDrag(const StringArray &files)* a *filesDropped(const StringArray& files, int x, int y)*. Těla těchto metod byla přepsána pro potřeby programu, ovšem hlavičky těchto funkcí zůstaly zachovány. Důležitým prvkem těchto funkcí je pole řetězců (StringArray), které je posíláno při volání těchto funkcí jako jejich parametr v hlavičce funkce. Toto pole řetězců obsahuje absolutní cestu k souborům, které byly vybrány a stiskem myši přesunuty do okna zásuvného modulu.

V těle metody *filesDropped(const StringArray &files, int x, int y)* je volána metoda *isInterestedInFileDrag(const StringArray &files)*, která zkoumá validitu formátu zvukového souboru, tedy jestli se jedná o soubor s příponou .wav nebo .aiff. Pokud je tato podmínka splněna, vrátí metoda návratovou hodnotu *true*, v opačném případě hodnotu *false*. Na základě této hodnoty je pak soubor buďto načten pomocí principu popsaného v kapitole 4.1, nebo je soubor přeskočen. Po tomto procesu nastává posun v poli s absolutními cestami vybraných souborů, dokud se v tomto poli nedostaneme na konec.

Pro lepší pochopení vyhledejte přílohu A.4, která obsahuje zdrojový kód metody *isInterestedInFileDrag(const StringArray &files)* a *filesDropped(const StringArray & files, int x, int y)*. [25]

4.3 Ukládání obsahu do vyrovnávací paměti

Aby bylo možné uložit obsah jednotlivých granulí i při případné změně vstupního souboru, je nutné abychom použili vyrovnávací paměti (*angl. buffer*). Každý kanál tak bude obsahovat sled těchto pamětí, ze kterých bude na základě ostatních parametrů granulární syntézy čten jejich obsah.

Projekt pro práci s vyrovnávacími pamětmi nahrává validní zvukový soubor (ve formátu .wav nebo .aiff) do hlavní vyrovnávací paměti v podobě vzorků. Pro reprezentaci granule je vytvořena další vyrovnávací paměť, která je ovšem svojí velikostí mnohem menší. V obou případech se jedná o objekt *AudioSampleBuffer*. [27]

Ve výsledném programu uživatel vybere místo počátku granule a nastaví její dobu trvání. Doba granule nastavená uživatelem je od počáteční pozice přepočtena na počet vzorků, které budou kopírovány z hlavní vyrovnávací paměti s uloženým vstupním souborem od pozice, kterou uživatel zadal. Pro jednoduchost tohoto samostatného projektu jsou zde hodnoty pevně dané.

Postup nahrání souboru je obdobný tomu popsanému v kapitole 4.1. Opět je vyvoláno dialogové okno s možností výběru souborů s příponami .wav a .aiff, je vytvořen čítač typu *AudioFormatReader,* který přečte soubor a nahraje navzorkovanou audio stopu do vyrovnávací paměti, která je objektem typu *AudioSampleBuffer*. Před samotným přečtením však musí být vyrovnávací paměti nastavena velikost, čítač proto musí paměti předat informace o počtu kanálů vstupního souboru a o celkovém počtu vzorků, které budou po tomto nastavení přečteny. [18] [20]

Pro vyrovnávací paměť granule je velikost této paměti pro jednoduchost nastavena pevně, velikost však bude závislá na době trvání granule. Kopírování obsahu je nutné dělat přes počet kanálů vstupního souboru a je nutné znát pozici, ze které chceme kopírovat. Také je potřebné znát množství vzorků, které chceme překopírovat, což se váže na již zmíněnou dobu trvání granule. Pro zjednodušení projektu jsou opět tyto hodnoty v projektu pevně dané, proto přílohy A.5 a A.6 mají pouze informativní charakter.

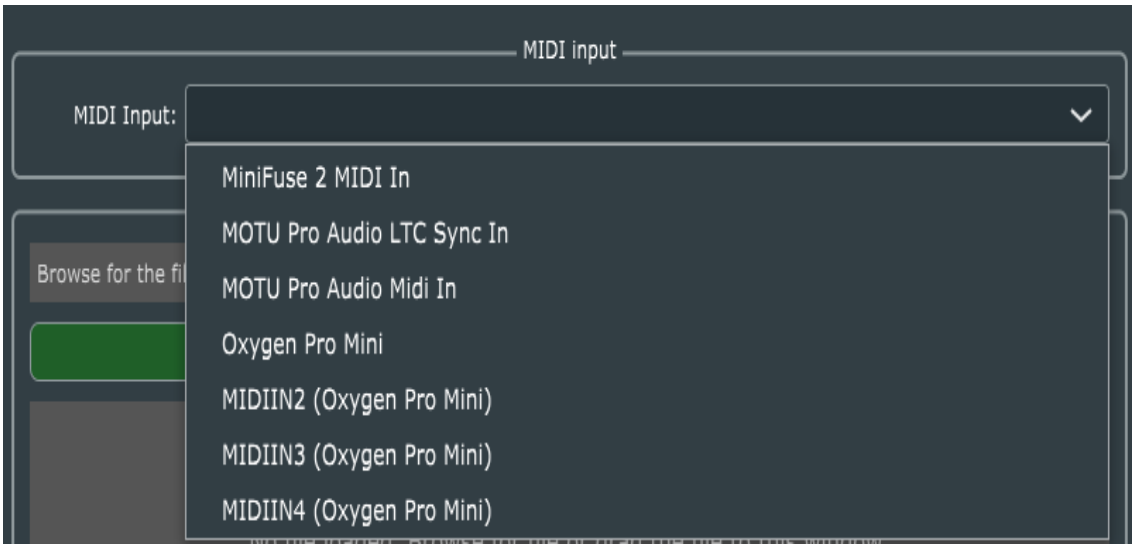
4.4 Komunikace skrze MIDI protokol

Aby bylo možné ovládat granulární syntézu pomocí kontroléru či kláves, je potřebné, aby program komunikoval skrze MIDI protokol. Pro tuto komunikaci obsahuje JUCE řadu tříd. Mezi ty, které byly v programu využity, náleží třída *MidiInputCallback* přijímající zprávy z fyzického zařízení a třída *MidiKeyboardStateListener*, která rozpoznává, které klávesy jsou zmáčknuty, na jakých kanálech a přiřadí k nim odpovídající MIDI zprávy. Důležité jsou také třídy *AudioDeviceManager,* pro výběr vstupního zařízení, a třída *MidiKeyboardComponent*, díky které je do okna zásuvného modulu přidána klaviatura, která kopíruje události z fyzického zařízení, a je schopna toto zařízení nahradit, pokud není přítomno. [15] [27] [28] [29]

Tento samostatný projekt je koncipován tak, že po stisku, či uvolnění klávesy zapisuje do okna s posuvníkem ze třídy *ComboBox* příslušné MIDI zprávy. Pro výsledný program bude použita reakce na zprávy *Nota zapnuta* a *Nota vypnuta*. V rámci samostatného projektu je do okna s posuvníkem kromě názvu zprávy tisknuta i informace o kanálech či notách, které se k této zprávě váží. [24]

Pro výsledný projekt bude z tohoto projektu čerpáno spíše okrajově. Důležité bude zejména zvolit vstupní zařízení a poskytnout uživateli klaviaturu ze třídy *MidiKeyboardComponent*, zajistit reakci na MIDI zprávy *Nota zapnuta* a *Nota vypnuta* a navázat tyto reakce na další činnost. Z tohoto důvodu nebude v příloze uveden zdrojový kód pro tuto problematiku, jelikož velká část tohoto kódu je spojena s výpisem MIDI zpráv a jiných detailů a nebude pro zbytek práce příliš relevantní. [28]

Na obrázku 4.2 je znázorněn seznam dostupných zařízení komunikujících skrze MIDI protokol, která mohou uživateli posloužit pro ovládání reprodukce.



Obrázek 4.2 Výběrový seznam pro volbu preferovaného zařízení, komunikujícího skrze MIDI protokol

5. REALIZACE NÁVRHU FUNKČNOSTI

V této kapitole bude rozebrán postup realizace finálního VST3 rozšíření vycházejícího z dřívějšího návrhu provedeném v rámci semestrální práce. V průběhu této realizace došlo v určitých částech programu ke komplikacím, které byly způsobeny zvolením špatného přístupu k realizaci dané části programu. Tyto problémy se tak negativně podepsali na celkovém výsledku práce, a to zejména z časového hlediska.

Díky řešení těchto problémů nebylo z časových důvodů možné zjistit, zda nedochází v programu k úniku paměťových bloků, tedy zda všechny alokované objekty jsou také správně de-alokovány a dochází k uvolnění paměti. Se zkracováním časového intervalu pro řešení práce rostla také nutnost programovat jednodušeji na úkor paměťové zátěže programu. Výsledný program je tak paměťově náročnější, než by musel a dochází zde v určitých částech k nedodržení kultury kódu, konkrétně ke kombinování českých a anglických názvů pro některé funkce a proměnné. Největším nedostatkem je však mírný odklon od návrhu funkčnosti. Z časových důvodů nebylo možné implementovat funkci, která dá uživateli možnost odstranit granule z daného kanálu. Dále nebylo implementováno vykreslení časového úseku granule ve zvukovém průběhu. Tuto absenci je ovšem možné obhájit nízkým rozlišením zhoršujícím se s rostoucí délkou vstupního souboru. Takovýto úsek by tedy nemusel být ve zvukovém průběhu povšimnutelný. Dále bylo odstraněno tlačítko pro vyvolání informačního okna a do GUI byla přidána třetí sekce s informacemi a nastavením zvukového zařízení.

Samotná realizace vychází (až na výše zmíněné výjimky) z návrhu funkčnosti a kódového řešení, které je popsáno v předchozí kapitole. Dále bylo nutné sestavit objekty modelující vlastnosti jednotlivých kanálů a granulí, přiřadit těmto objektům patřičné parametry, vytvořit amplitudové obálky a aplikovat je na jednotlivé granule, provést vizualizaci toku zvuku jednotlivými kanály a rozdistribuovat zvuk na tyto kanály.

Dále budou rozebrány jednotlivé části realizace a přiblíženo programové řešení těchto částí. Konkrétní popis všech funkcí je uveden v přiloženém archivu ve zdrojových souborech.

5.1 Objekt granule, kanál a jejich provázání

Tyto dva objekty, též třídy, jsou základní stavební jednotkou celého návrhu a celého principu této práce. Jedná se o objekt granule s názvem *Grain* a objekt kanálu s názvem *Channel*.

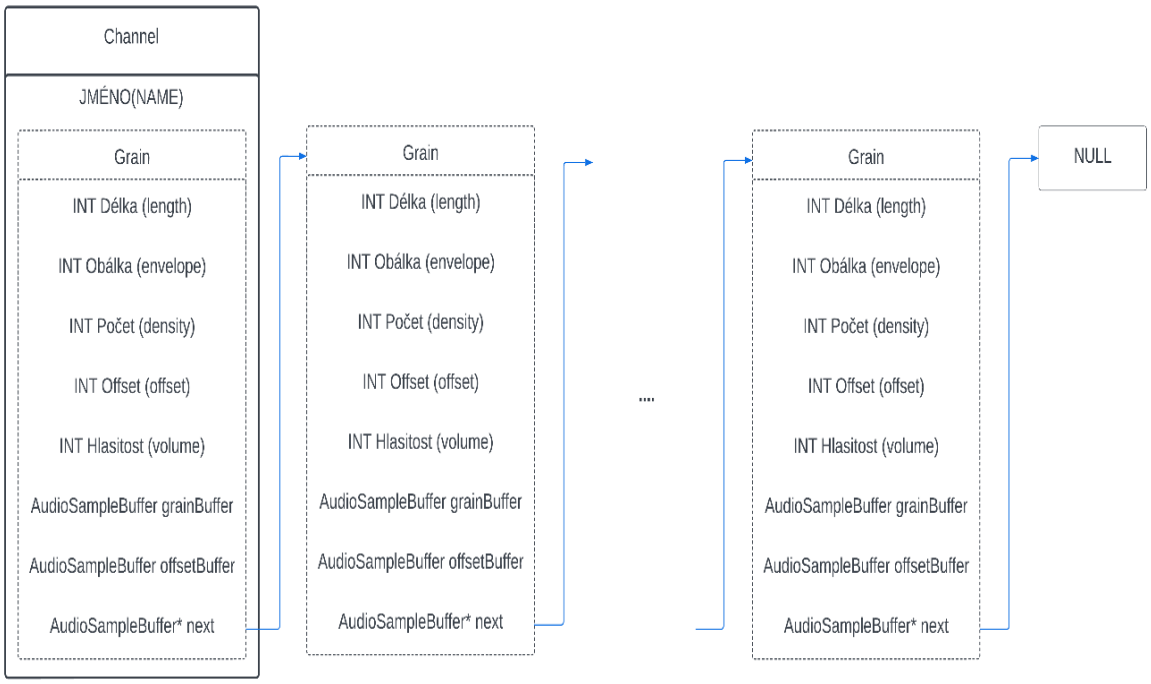
Oba tyto objekty jsou na sebe vzájemně navázány. Objektů *Channel* je celkem osm a každý z těchto objektů v sobě uchovává ukazatel na objekt *Grain*. Každý objekt *Grain* uchovává ukazatel na další objekt *Grain*, granule jsou tak řazeny do jednoduše vázaného seznamu, kde se objekt *Channel* chová jako hlava nebo též počátek seznamu. Strukturu tohoto provázání objektů přibližuje obrázek 5.1.

Objekt *Channel* obsahuje kromě ukazatele na objekt *Grain* také řetězec, který zde funguje jako jeho identifikátor. Objekt *Grain* obsahuje mimo zmíněný ukazatel také informaci o délce granule, její amplitudové obálce, počtu granulí v jedné sekundě, časovém offsetu a hlasitosti granule ve výsledném zvuku.

Dále obsahuje dva objekty *AudioSampleBuffer*. Do jednoho je uložena zvuková informace granule v podobě jednotlivých vzorků, druhý objekt nese informaci o časovém offsetu již tvořenou nulovými vzorky. Oba objekty jsou velmi důležité pro správnou funkčnost programu v reálném čase. [20]

Objekt *Grain* je oproti návrhu funkčnosti nepatrně upraven. Obsahuje informaci o hlasitosti granule ve výsledném zvuku, čímž může uživatel kontrolovat dynamiku zvuku v jedné sekundě, a docílit tak zajímavějších zvukových struktur. Druhá změna se týká délky časového offsetu, která byla zkrácena z pěti minut na minuty dvě.

Pro snadnější pochopení jsou v přílohách A.7 a A.8 uvedena kódová řešení obou těchto objektů.



Obrázek 5.1 Vizuální struktura jednoduše vázaného seznamu tvořeného objekty *Grain* a *Channel*

5.2 Realizace amplitudových obálek

Realizace trojúhelníkové (AR) a lichoběžníkové (ASR) amplitudové obálky byla jedna z částí, která zpozdila průběh této práce. Při realizaci bylo nutné opustit princip, který byl pro obálky využit a vydat se jiným směrem.

Původní princip vycházel z oficiální dokumentace frameworku JUCE, kde je uvedena třída *ADSR*. Jedná se o třídu, která zavádí jednoduchou ADSR obálku, tedy obálku s parametry *attack*, *decay*, *sustain* a *release*. Detailní popis této třídy instruuje uživatele k vytvoření objektu *ADSR*, nastavení vzorkovací rychlosti pro tento objekt, nastavení parametrů *ADSR* a k volání funkce *applyEnvelopeToBuffer(AudioBuffer<FloatType>& buffer, int startSample, int numSamples)* či funkce *getNextSample()*. [13]

Jelikož zvukový obsah granulí je uložen v objektech *AudioSampleBuffer*, byla první jmenovaná funkce přívětivější variantou. Třída *ADSR* obsahuje také funkce *noteOn()* a *noteOff(),* které spouští fázi *attack* a *release* dané obálky. Tyto dvě funkce byly navázány na stisk tlačítka Play Grain, viz obrázek 3.5. Při spuštění projektu však nedošlo k očekávanému výsledku. Se stiskem tlačítka začala fáze *attack*, *decay* a *sustain*, poslední jmenovaná fáze však trvala do té doby, dokud nedošlo k opětovnému stisknutí tlačítka Play Grain, kdy byla vyvolána funkce *noteOff()*.[13] [20]

Ani po delším testování a úpravách kódu nebylo docíleno správného chování programu, proto bylo nutné změnit přístup k realizaci této části programu.

Tato druhá koncepce opět vychází z oficiální dokumentace, konkrétně ze třídy *AudioBuffer*, která je rodičovskou třídou objektů *AudioSampleBuffer*, do kterých je ukládán zvukový obsah granule. Tyto objekty mohou volat funkci *applyGainRamp(int channel, int startSample, int numSamples, Type startGain, Type endGain)*. Po zavolání této funkce na daném kanále bufferu bude od zadané startovní pozice pro zadaný počet prvků provedeno přizpůsobení hlasitosti podle lineární funkce mezi počáteční a koncovou hlasitostí. Pomocí této funkce je tedy možné napodobit stejné vlastnosti, jako u výše zmíněné třídy *ADSR*. [14] [20]

Nevýhodou tohoto principu je jeho aplikace na lichoběžníkovou (ASR) obálku. Zde je fáze *attack* tvořena 30 % vzorků dané granule, fáze *sustain* dalšími 40 % vzorků a fáze *release* zbylými 30 %. Při zjišťování počtu vzorků, na které bude lineární přizpůsobení hlasitosti provedeno, je nutné při výpočtu zaokrouhlovat na celý počet vzorků, čímž se do výsledku vnáší nepatrná odchylka od očekávaného průběhu. Jednotlivé úseky této obálky tak mohou být o vzorek kratší nebo o vzorek delší. Vzhledem k délce samotné granule však není tento jev při reprodukci granulí jednoznačně rozpoznatelný a přispívá tomu i vzorkovací kmitočet 48 kHz. Při velmi nízkém vzorkovacím kmitočtu a znalosti zvukového obsahu granule by bylo pravděpodobně možné tuto chybu rozpoznat. Realizace amplitudové obálky tvaru Gaussovi funkce vychází z podobného konceptu jako funkce *applyGainRamp*.

Gaussova funkce je dána vztahem [1] :

$$f(x) = \mathrm{a} \cdot e^{\left(-\frac{(x-b^2)^2}{2c^2}\right)}, \tag{5.1}$$

kde f(x) představuje průběh obálky v časové oblasti, parametr a = 1, parametr b je časovým údajem maxima této křivky a parametr c určuje šířku této zvonovité křivky. Zvukový obsah objektu *AudioSampleBuffer* uchovávajícího granuli je po jednotlivých vzorcích násoben s průběhem této obálky čímž dochází ke změně jednotlivých vzorků a tím ke změně jejich hlasitosti. [20]

5.3 Vytvoření objektů

Objekty Channel reprezentující jednotlivé kanály zvukového systému jsou vytvořeny staticky při spuštění VST3 pluginu. Poté, co uživatel nahraje zvukový soubor, vybere vlastnosti granule a zatrhne, na které kanály má být granule přidána, musí být stisknuto tlačítko *Create Grain*, které spouští funkci *createGrain()*.

Tělo funkce *createGrain()* je tvořeno podmíněnými výrazy, kdy je kontrolováno zatrhnutí objektů *ToggleButton*. Jedná se o zaškrtávací boxy pro jednotlivé kanály, pokud je box pro daný kanál zaškrtnut, je volána funkce *append(Grain **head)*, která na konec jednoduše vázaného seznamu připojí objekt *Grain*. [34]

V těle funkce *append(Grain **head)* dochází k vytvoření dvou objektů *AudioSampleBuffer*. První z nich je poslán jako parametr funkci *fillBuffer(juce::AudioSampleBuffer* buffer)*, ve které dojde k překopírování zvukového obsahu granule ze vstupního souboru. Dále je tento objekt poslán jako parametr do funkce *adsrSettings(juce::AudioBuffer<float>& buffer)*, kde je na zvuková data aplikována amplitudová obálka. Druhý objekt *AudioSampleBuffer* je poslán jako parametr funkci *addZeroesToBuffer(juce::AudioSampleBuffer& buffer)*, která naplní buffer nulovými vzorky, čímž je při reprodukci simulován časový offset daného toku granulí. Dále je dynamicky alokován objekt *Grain*, v jehož konstruktoru jsou předány objektu hodnoty jednotlivých posuvníků nastavujících parametry granulí a oba výše zmíněné buffery. Takto vytvořený objekt je poté přidán na konec jednoduše vázaného seznamu, jehož počátek určuje parametr *head* funkce *append(Grain **head)*. [14] [20]

5.4 Realizace vizualizace toku zvuku

Cílem vizualizace zvukového toku je přiblížit uživateli, jakým kanálem je v daný okamžik zvuk zrovna reprodukován. Mimo tuto vlastnost má také vizualizace za úkol přiblížit jednotlivé algoritmy, podle kterých reprodukce probíhá. Uživatel má možnost si před samotnou reprodukcí algoritmy prohlédnout a dle vlastního uvážení jeden z algoritmů zvolit.

Základem celé vizualizace je osm objektů *TextButton* umístěných v GUI. Těchto osm objektů mění barvu svého pozadí v čase na základě čtení dat z pole, ve kterém je uloženo pořadí kanálů pro jednotlivé algoritmy. [30]

Po stisku tlačítka SHOW PATTERN je vše řízeno časovačem ze třídy *Timer*, konkrétně metodou *startTimer(int intervalInMiliseconds)*. Při stisku tlačítka je zjištěno, který algoritmus byl vybrán a spuštěn časovač, který funguje tak, že neustále nastavuje nový časový cyklus s délkou *time* v milisekundách. Po ukončení každého cyklu je vyvolána funkce *timerCallback()*. Tato funkce volá další funkci (*showPattern()*), která se stará o přebarvení pozadí na očekávaném tlačítku. Podle obsahu pole na daném indexu je pomocí dvou příkazů *switch* změněno pozadí prvku, který bude v dalším cyklu ukazovat pozici zvuku v systému a pozadí prvku, který již pozici ukazoval, je vráceno do původní podoby. Nakonec dochází k inkrementaci indexu pole, aby nedocházelo ke čtení ze stále stejné pozice. [33]

Při přehrávání zvuku funguje tento princip podobně, spouštěčem je však klávesa vysílající MIDI zprávu *Nota zapnuta*, která spouští časovač, nikoliv tlačítko SHOW PATTERN.

Nedostatkem tohoto řešení je fakt, že nejdříve musí proběhnout celý jeden cyklus od spuštění časovače, než je vyvolána funkce *timerCallback()* a funkce *showPattern()*, kdy dojde k první vizuální změně. [33]

Tento jev se negativně podepsal i na reprodukci granulární syntézy osmikanálovým systémem, jelikož zvukový výstup musel být taktéž pozdržen o daný časový úsek, aby byl proces synchronizován s vizualizací. Po stisku klávesy tedy dochází k prodlevě, po které začne systém reprodukovat výsledný produkt společně s vizuální informací o toku zvuku.

5.5 Realizace syntézy zvuku a její distribuce

Realizace této části byla jednoznačně nejnáročnější částí celé práce. Jelikož se jedná o dosti specifickou problematiku, na kterou nelze najít odpovědi ani na oficiálním fóru frameworku JUCE, byla pro tuto část implementace zvolena metoda pokusu a omylu. Důležitost zde byla kladena na implementaci jednotlivých toků granulí nejdříve na jediném kanále, který byl distribuován do stereo zvukového systému. Po implementaci této problematiky bylo pak nutné daný přístup upravit i pro zbylé kanály a umožnit distribuci zvuku do navrhnutého osmikanálového systému.

Princip granulární syntézy, jak už bylo řečeno, spočívá v neustálém se opakování jednotlivých granulí od stisku klávesy až po její uvolnění. Při snaze efektivně implementovat tento princip byl zvolen přístup, který měl za cíl každý buffer se zvukovou informací opakovat a vytvořit tak samostatný proud granulí nezávisle na ostatních granulích. Tento princip měl efektivně pracovat s procesorem a umožnit, aby toky granulí na jednotlivých kanálech běžely nezávisle na sobě a byly slučovány v komplexní zvuk.

Tento přístup pracoval s objektem *ThreadPool*, který definuje počet vláken, na kterých měla být realizována funkce cyklení jednotlivých přehrávaných granulí. Po inicializaci objektu *ThreadPool* byla se stiskem klávesy volána funkce *startThread()*, která připraví volné vlákno k procesu a poté volá funkci *run()*. [32] [31]

Tato funkce musela být předefinována tak, aby spustila funkci *getNextAudioBlock(const juce::AudioSourceChannelInfo& bufferToFill)*, kde buffer *bufferToFill* je výstupním buffrem celého systému a je na něj kopírován výsledný zvuk, který je dále reprodukován. Při spuštění programu byl tento přístup testován na jednom z osmi dostupných kanálů. Při testování tohoto principu byla vytvořena granule, přidána na zvolený kanál a byla spuštěna reprodukce, při které došlo k očekávanému výstupu, tedy bylo možné slyšet zatím jediný tok opakující se granule. K chybě ovšem došlo v okamžiku, kdy na zvolený kanál byla přidána další granule, která vyvolala vytvoření nového procesu na novém vlákně. Při tomto pokusu o vytvoření a volání nového procesu byl celý program přerušen s chybou neoprávněného přístupu do paměti. S touto chybou nebylo možné dále pokračovat v implementaci a bylo nutné zjistit, proč k dané chybě dochází. To se ovšem nepovedlo i navzdory snaze o řízení procesů či mapování práce s pamětí pomocí debuggeru. Tento přístup tak musel být opuštěn, což posunulo realizaci této části práce zpět do výchozí pozice. [21]

Nový přístup, se kterým je problematika složení jednotlivých toků granulí dohromady ve výsledný zvuk řešena, je založen na sčítaní vzorků jednotlivých granulí. V souboru *midiKomunikace.h* byl definován objekt *AudioSampleBuffer*, na který jsou přičítány vzorky jednotlivých granulí na daném kanále. I zde byl princip nejdříve implementován pro stereo reprodukci a pouze s jedním objektem *Channel*. Při stisku klávesy došlo k postupnému průchodu jednoduše vázaným seznamem granulí a ke čtení obsahu těchto granulí, kdy u každé granule došlo k přičtení jednotlivých vzorků granule ke vzorkům definovaného objektu *AudioSampleBuffer*. Velikost tohoto objektu ve vzorcích byla o sekundu větší než délka maximálního časového offsetu granule. Každá granule byla na tento objekt přičtena několikrát podle nastaveného časového offsetu tak, aby byl dodržen princip lineárního řazení granulí. Ve funkci *getNextAudioBlock(const juce::AudioSourceChannelInfo& bufferToFill)* došlo postupně ke kopírování vzorků objektu *AudioSampleBuffer* na oba kanály výstupního bufferu *bufferToFill* s tím, že pokud byl počet vzorků objektu *AudioSampleBuffer* vyčerpán, došlo k posunu na pozici o jednu sekundu zpět. Tímto bylo zajištěno neustálé cyklení i po uplynutí maximálního časového offsetu granulí a bylo docíleno požadovaného jevu, tedy složení všech toků granulí na daném kanále do výsledného zvuku. [20] [21]

Aby bylo možné tento princip aplikovat na osmikanálový zvukový systém bylo nutné definovat dalších sedm objektů *AudioSampleBuffer*. Společně s již definovaným objektem tyto objekty reprezentují jednotlivé kanály zvukového systému. Do každého z těchto objektů jsou přičteny vzorky granulí uložených na patřičném kanále již zmíněným principem. [20]

Framework JUCE při spuštění programu definuje objekt *AudioDeviceManager*, který jako výstupní zařízení prohlásí to zařízení, které má operační systém v daný moment nastaveno jako výchozí. Tento objekt je tedy při tvorbě GUI přepsán a pro uživatele je pomocí objektu *AudioDeviceSelectorComponent* vytvořeno výběrové menu, kde lze zvolit správné výstupní zařízení, v případě této práce se jednalo o osmikanálovou zvukovou kartu. Díky přepsání původního objektu *AudioDeviceManager* je nastaven výstup VST3 pluginu na osmikanálový, tedy výstupní buffer funkce *getNextAudioBlock(const juce::AudioSourceChannelInfo& bufferToFill)* se stává osmikanálovým. Kopírováním zvukových dat objektů *AudioSampleBuffer* na buffer *bufferToFill* je docíleno toho, že zvuk bude reprodukován na kanále, na který jsou data kopírována. Toto kopírování dat je řízeno jedním z algoritmů, který si uživatel dle svého uvážení dříve vybral. [15] [16] [20] [21]

Reprodukce výsledného zvuku není možná, pokud nebude připojeno zařízení s dostatečným počtem výstupů, minimálně osmi. V takovémto případě si může uživatel přehrát zvukový soubor, vytvořit jednotlivé granule, zobrazit algoritmus, kterým by reprodukce probíhala, ale výsledný zvuk přehrán nebude, jelikož jednotlivé algoritmy jsou sestaveny právě pro osmikanálovou reprodukci. Pokud by v programu toto omezení nebylo implementováno, došlo by při výběru nekompatibilního zařízení k přístupu na nedefinovaná místa v paměti počítače, a tedy k přerušení programu s chybou.

6. TESTOVÁNÍ PLUGINU A JEHO NEDOSTATKY

V této kapitole jsou rozebrány metody, kterými byl VST3 plugin testován a uvedeny největší nedostatky jeho koncové verze.

6.1 Testování pluginu

Testování VST3 pluginu probíhalo ve dvou fázích. V první fázi byl plugin průběžně testován při jednotlivých sestaveních programu, kdy byly odhalovány kritické nedostatky a chyby, které byly okamžitě řešeny a opravovány. Tato fáze probíhala postupně s rozšiřováním kódu o nové funkce, kdy při implementaci nové funkce byla testována funkčnost programu jako celku s důkladnějším zaměřením především na nově implementovanou funkci. Postupně tedy bylo testováno načítání souborů a funkčnost obsluhy přehrávače souborů, následně tvorba jednotlivých granulí, kdy bylo testováno správné přiřazení parametrů těmto objektům, a zejména správná alokace a přiřazení adresy těchto objektů v paměti. Dále bylo testováno vykreslování jednotlivých algoritmů pro reprodukci pomocí tlačítka SHOW PATTERN, reprodukce samotná, a nakonec testování reprodukce společně s vykreslováním toku zvuku osmikanálovým systémem. První fáze testování byla (až do fáze testování samotné reprodukce) provedena ve stereofonním systému.

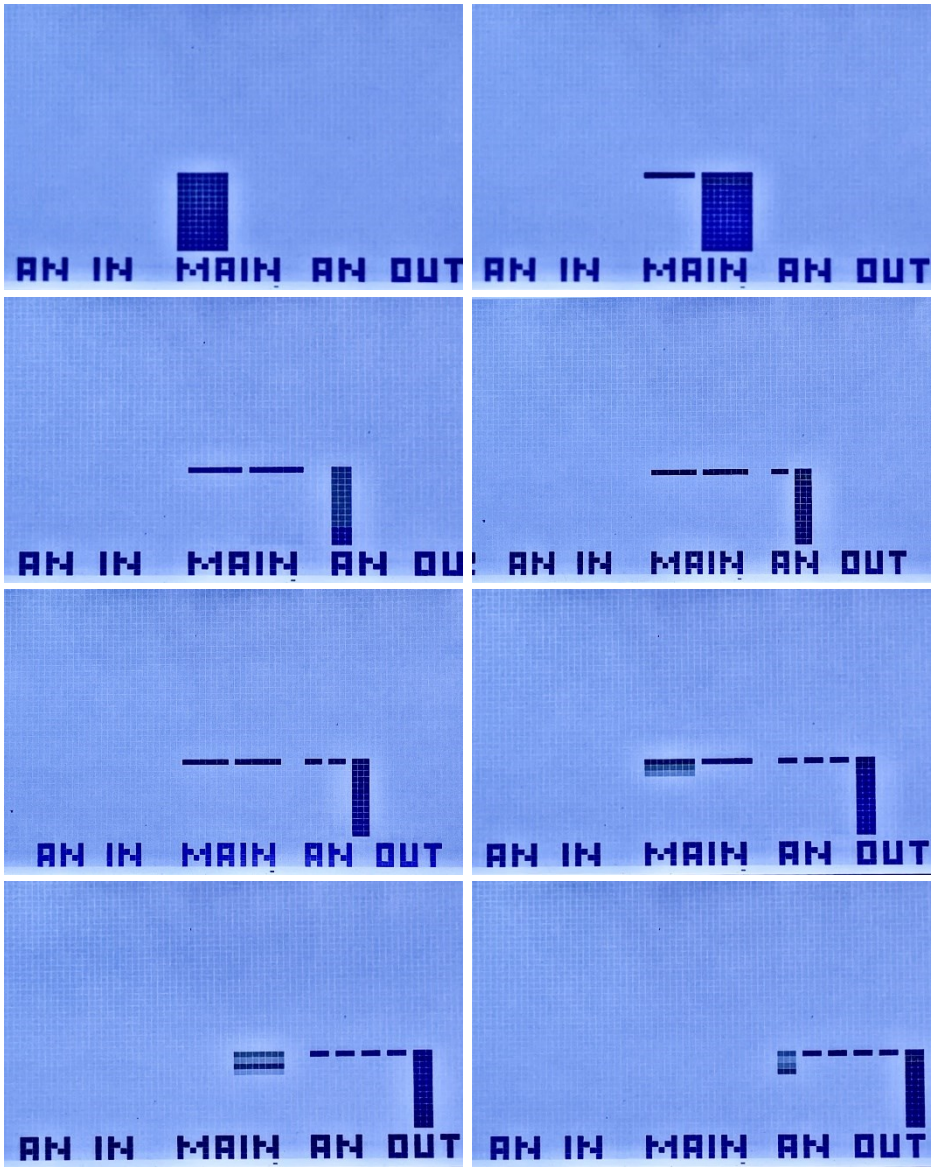
Pro testování posloužila dvě zařízení se sluchátkovým výstupem, konkrétně přenosný počítač s operačním systémem Windows 8.1 a stolní počítač s operačním systémem Windows 10. Plugin byl testován aplikací *AudioPluginHostDebug.exe* frameworku JUCE a také třemi DAW systémy, konkrétně se jednalo o *FL Studio 21*, *Reaper* a *Steinberg Cubase*.

Pro testování samotné reprodukce a reprodukce s doprovodem vizualizace toku zvuku systémem byl použit pouze stolní počítač se zvukovou kartou *MOTU UltraLight AVB* zapůjčenou z Janáčkovy akademie múzických umění. Testování probíhalo na sluchátkovém výstupu s vizuální kontrolou distribuce zvuku na zbylé výstupy této karty. Přenosný počítač k těmto účelům nebylo možné použít skrze nesplňující systémové požadavky pro zapůjčenou zvukovou kartu.

Druhá fáze testování proběhla v prostorách Janáčkovy akademie múzických umění, kde bylo možné vyzkoušet funkčnost výsledného pluginu na místním osmikanálovém systému, který se skládal z osmi aktivních monitorů *Genelec 8030*. Pro testování byly použity nejdříve místní počítače s operačním systémem Windows 10, na kterých byl plugin spouštěn v DAW systému *Reaper*. Na těchto počítačích však došlo k problémům se skenováním pluginu ze složky s ostatními soubory formátu VST3. V tomto případě se s největší pravděpodobností jednalo o problém konkrétní verze DAW *Reaper*, jelikož plugin byl dříve testován na vícero zařízeních v různých DAW, kde žádný takovýto problém při skenování souborů nenastal.

Zbytek testování tedy proběhl na osobním stolím počítači, který byl se systémem aktivních monitorů propojen pomocí zapůjčené zvukové karty MOTU. Výstup tohoto testu odpovídal programové realizaci, tedy bylo možné nahrát a přehrát zvukový soubor, vytvořit granuli z určité pozice ve vstupním souboru, přidat tuto granuli na zvolené kanály, nechat si vykreslit zvolený algoritmus reprodukce, započít reprodukci stisknutím klávesy a poslechnout si výslednou distribuci zvuku do celého systému dle zvoleného reprodukčního algoritmu včetně vizuálního zachycení této distribuce.

Na obrázku níže je zachycen průchod zvuku jednotlivými kanály reprodukčního systému při testování kruhového reprodukčního algoritmu.



Obrázek 6.1 Záznam displeje zvukové karty demonstrující přítomnost signálu na výstupech této karty při reprodukci

6.2 Nedostatky odhalené při finálním testování

Kromě již zmíněného nedostatku, tedy odepření uživateli možnosti odebrat jednotlivé granule z vybraného kanálu, se při testování objevily další nedostatky konečné podoby VST3 pluginu. Tyto nedostatky neměly přímo vliv na funkčnost a zvukový výstup pluginu, jednalo se spíše o estetické nedostatky spojené se zobrazováním a chováním pluginu.

Prvním nedostatkem je časová prodleva, která se z neznámých důvodů projeví při načítání zvukového souboru. V momentě, kdy uživatel vybere zvukový soubor, se kterým chce v pluginu dále pracovat, dojde až po okamžik vykreslení zvukového průběhu k prodlevě v délce trvání přibližně čtyři sekundy. Tento nedostatek sice na zvukový výsledek, pro který byl plugin navrhován a konstruován, nemá žádný vliv, ale na atraktivitě tento jev nástroji určitě nepřidává.

Druhým nedostatkem, který se při testování projevil, je problém s vykreslováním indikujícím tok zvuku jednotlivými kanály. Zde je problém způsobený samotnou koncepcí programu, na které je vykreslování toku zvuku kanálem postaveno. Samotné vykreslování je spuštěno paralelně se spuštěním reprodukce. Problémem zde je, že je vykreslování řízeno časovačem, který první vykreslení provede až uplynutí doby nastavené časovači. V momentě, kdy by došlo k obnovení cyklu reprodukčního algoritmu, by za normálních okolností došlo ke zpoždění vykreslení právě o daný časový úsek. Proto je v programu volán nový cyklus vykreslování vždy jeden krok před obnovou reprodukčního algoritmu. Následkem tohoto předčasného volání je stav, kdy dochází ke konfliktu mezi nově započatým, a právě končícím vykreslováním. Chování programu při tomto konfliktu není předvídatelné a dochází tak k tomu, že v určitých případech plugin správně indikuje přítomnost zvuku na daném kanále, v opačném případě však k vykreslení nedochází.

Problém s vykreslováním až po uplynutí prvního cyklu časovače negativně ovlivňuje také samotnou reprodukci, která díky tomuto jevu musí být po stisku klávesy pozdržena o stejný časový okamžik. Při současné koncepci pluginu zvuk putuje z určitého kanálu na jiný s intervalem jedné sekundy, o tento interval tak musí být reprodukce pozdržena.

Pokud by však došlo k prodloužení intervalu přechodu mezi jednotlivými kanály, bylo by nutné navýšit tuto prodlevu pro správné chování vizualizace toku zvuku, což by bylo z uživatelského hlediska dosti nepraktické.

ZÁVĚR

Cílem této práce bylo navrhnout a realizovat experimentální software pro distribuci granulární syntézy do prostorového zvuku. Návrh, realizace a výsledná podoba předmětu této práce byla postavena nejen na teoretických poznatcích o granulární syntéze a vícekanálové reprodukci, ale také na dispozicích frameworku JUCE, ve kterém byl software vytvořen. Ač je tento framework postaven na programovacím jazyku C++, nebylo by možně výsledný produkt vytvořit bez důkladného prostudování knihoven, uvedených v oficiální dokumentaci tohoto frameworku.

Po prostudování teoretických poznatků byl výstupní software navržen jako VST3 plugin, zejména pro svou možnost importovat tento formát do systémů DAW. Funkční návrh tohoto pluginu byl postaven na teoretických poznatcích o granulární syntéze s přihlédnutím na uživatelskou vstřícnost prostředí. Hlavním cílem však zůstala distribuce výsledného zvuku do vícekanálového systému, aby došlo ke splnění tématu této práce.

Výsledkem je tedy VST3 plugin, který je možné importovat do systémů DAW. Plugin reaguje na zprávy *Nota zapnuta* a *Nota vypnuta* protokolu MIDI, které mohou být vysílány skrze připojené zařízení a které ovládají reprodukci osmikanálovým systémem. Díky komplikacím, které v této práci při realizaci nastaly, byl v konečné fázi této práce nutný odklon od funkčního návrhu. Dle tohoto návrhu měl mít uživatel VST3 pluginu možnost odstraňovat jím vybrané granule z daných kanálů. V konečné fázi této práce byla nutná implementace funkce distribuující zvuk do vícekanálového systému na úkor této funkce, a to zejména pro splnění tématu této práce.

Celková implementace výsledného pluginu má několik nedostatků. Kódově je plugin řešen tak, aby splňoval zadané téma i za cenu nepříliš ideálního hospodaření s pamětí. Samotnou paměť nebylo z časových důvodů možné otestovat, takže není vyloučeno, že program nezanechává v paměti alokované bloky. Testování pluginu odhalilo zejména estetické nedostatky spojené s vizualizací toku zvuku jednotlivými kanály, pro jejichž odstranění by muselo dojít ke změně celkové koncepce vizualizace. Tyto nedostatky by se tak mohly stát společně doplněním funkčního návrhu o nové funkce předmětem případné diplomové práce.

I přes výše popsané nedostatky a problémy, které práci prováděly, je možné konstatovat, že existuje způsob, jímž lze granulární syntézu distribuovat do vícekanálového reprodukčního systému za použití frameworku JUCE pro tento účel.

LITERATURA

[1] Gaussian function. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2023 [cit. 2023-05-14]. Dostupné z: https://en.wikipedia.org/wiki/Gaussian_function

[2] ŽURKOVÁ, Darina, 2021. Granulární princip a jeho uplatnění v umělecké kompoziční praxi – především se zaměřením na zvukový design z perspektivy témbru [Granular principle and its application in artistic composition practice – especially with a focus on sound design from the perspective of timbre]. Brno. Disertační práce. 165 s. Janáčkova akademie múzických umění v Brně. Školitel prof. Ing. MgA. Ivo Medek, PhD.

[3] SYROVÝ, Václav, 2013. Hudební akustika. 3., dopl. vyd. Praha: Akademie múzických umění. Akustická knihovna Zvukového studia Hudební fakulty AMU. ISBN 978-80-7331-297-8.

[4] EVEREST, F. Alton a Ken C. POHLMANN. Master handbook of acoustics. Seventh edition. New York: McGraw Hill, [2022]. ISBN 978-126-0473-599.

[5] ROADS, Curtis. Microsound. Cambridge, Mass.: MIT Press, c2001. ISBN 0-262-18215-7.Směrnice č. 72/2017

[6] Octophonic sound. Wikipedia [online]. 2022, 30.9.2022 [cit. 2022-11-25]. Dostupné z: https://en.wikipedia.org/wiki/Octophonic_sound

[7] Octophony, by Daniel Laberge [online]. [cit. 2022-11-25]. Dostupné z: https://www.daniellaberge.net/music/octophony/octophony1.htm

[8] RUSS, Martin. Sound synthesis and sampling. Oxford: Focal Press, 1996. Music technology series. ISBN 0-240-51429-7.

[9] HOLMAN, Tomlinson. Surround sound: up and running. 2nd ed. Amsterdam: Focal Press, 2008. ISBN 978-0-240-80829-1.

[10] MIDI MANUFACTURERS ASSOCIATION. The Complete MIDI 1.0 Detailed Specification: Incorporating all Recommended Practices, document version 96.1 [PDF]. Third edition. Los Angeles, CA: The MIDI Manufacturers Association, c1995-2006 a 2014 [cit. 2022-11-18]. ISBN 90632-3173. Dostupné z: https://www.midi.org/downloads?task=callelement&format=raw&item_id=92&element=f85c494b-2b32-4109-b8c1-083cca2b7db6&method=download

[11] HUBER, David Miles. The MIDI Manual: A Practical Guide to MIDI in the Project Studio. 3rd edition. United States America: Foca Press, c2007. ISBN 978-0-240-80798-0.

[12] VST 3 Developer Portal: Welcome to the world of VST 3 [online]. c2022 [cit. 2022-11-19]. Dostupné z: https://developer.steinberg.help/display/SDH/Steinberg+Developer+Resource

Knihovny JUCE

[13] JUCE dokumentace ke třídě ADSR. *JUCE* [online]. [cit. 2023-05-18]. Dostupné z: https://docs.juce.com/master/classADSR.html

[14] JUCE dokumentace ke třídě AudioBuffer. *JUCE* [online]. [cit. 2023-05-18]. Dostupné z: https://docs.juce.com/master/classAudioBuffer.html

[15] JUCE dokumentace ke třídě AudioDeviceManager. *JUCE* [online]. [cit. 2023-05-18]. Dostupné z: https://docs.juce.com/master/classAudioDeviceManager.html

[16] JUCE dokumentace ke třídě AudioDeviceSelectorComponent. *JUCE* [online]. [cit. 2023-05-19]. Dostupné z: https://docs.juce.com/master/classAudioDeviceSelectorComponent.html

[17] JUCE dokumentace ke třídě AudioFormatManager. *JUCE* [online]. [cit. 2023-05-18]. Dostupné z: https://docs.juce.com/master/classAudioFormatManager.html

[18] JUCE dokumentace ke třídě AudioFormatReader. *JUCE* [online]. [cit. 2023-05-18]. Dostupné z: https://docs.juce.com/master/classAudioFormatReader.html

[19] JUCE dokumentace ke třídě AudioFormatReaderSource. *JUCE* [online]. [cit. 2023-05-18]. Dostupné z: https://docs.juce.com/master/classAudioFormatReaderSource.html

[20] JUCE dokumentace ke třídě AudioSampleBuffer. *JUCE* [online]. [cit. 2023-05-18]. Dostupné z: https://docs.juce.com/master/group__juce__audio__basics-buffers.html

[21] JUCE dokumentace ke třídě AudioSource. *JUCE* [online]. [cit. 2023-05-19]. Dostupné z: https://docs.juce.com/master/classAudioSource.html

[22] JUCE dokumentace ke třídě AudioThumbnail. *JUCE* [online]. [cit. 2023-05-18]. Dostupné z: https://docs.juce.com/master/classAudioThumbnail.html

[23] JUCE dokumentace ke třídě AudioTransportSource. *JUCE* [online]. [cit. 2023-05-18]. Dostupné z: https://docs.juce.com/master/classAudioTransportSource.html

[24] JUCE dokumentace ke třídě ComboBox. *JUCE* [online]. [cit. 2023-05-18]. Dostupné z: https://docs.juce.com/master/classComboBox.html

[25] JUCE dokumentace ke třídě FileDragAndDropTarget. *JUCE* [online]. [cit. 2023-05-18]. Dostupné z: https://docs.juce.com/master/classFileDragAndDropTarget.html

[26] JUCE dokumentace ke třídě FileChooser. *JUCE* [online]. [cit. 2023-05-18]. Dostupné z: https://docs.juce.com/master/classFileChooser.html

[27] JUCE dokumentace ke třídě MidiInputCallback. *JUCE* [online]. [cit. 2023-05-18]. Dostupné z: https://docs.juce.com/master/classMidiInputCallback.html

[28] JUCE dokumentace ke třídě MidiKeyboardComponent. *JUCE* [online]. [cit. 2023-05-18]. Dostupné z: https://docs.juce.com/master/classMidiKeyboardComponent.html
[29] JUCE dokumentace ke třídě MidiKeyboardStateListener. *JUCE* [online]. [cit. 2023-05-18]. Dostupné z: https://docs.juce.com/master/classMidiKeyboardState_1_1Listener.html
[30] JUCE dokumentace ke třídě TextButton. *JUCE* [online]. [cit. 2023-05-19]. Dostupné z: https://docs.juce.com/master/classTextButton.html
[31] JUCE dokumentace ke třídě Thread. *JUCE* [online]. [cit. 2023-05-19]. Dostupné z: https://docs.juce.com/master/classThread.html
[32] JUCE dokumentace ke třídě ThreadPool. *JUCE* [online]. [cit. 2023-05-19]. Dostupné z: https://docs.juce.com/master/classThreadPool.html
[33] JUCE dokumentace ke třídě Timer. *JUCE* [online]. [cit. 2023-05-19]. Dostupné z: https://docs.juce.com/master/classTimer.html
[34] JUCE dokumentace ke třídě ToggleButton. *JUCE* [online]. [cit. 2023-05-18]. Dostupné z: https://docs.juce.com/master/classToggleButton.html

SEZNAM SYMBOLŮ A ZKRATEK

Zkratky:

DAW	Digital Audio Workstation
GUI	Grafické uživatelské rozhraní
MSB	Nejvíce významný bit
MIDI	Musical Instrument Digital Interface
VUT	Vysoké učení technické v Brně

SEZNAM PŘÍLOH

Příloha A - Kódová řešení částí projektů pro SP

A.1 Kód pro načtení zvukového souboru

```
/*******************************************************************
*     Funkce prepareForPlaying pro načtení zvukového souboru
*/

void BrowsefileAudioProcessorEditor::prepareForPlaying(juce::File file)
{
    // registrace formátů .wav a .aiff
    formatManager.registerBasicFormats();

    // kontrola validity souboru
    if (file != juce::File{})
    {
        infoLabel.setText(file.getFullPathName(),
                          juce::dontSendNotification);

        // vytvoření čítače AudioFormatReader
        auto* reader = formatManager.createReaderFor(file);

        // testování validnosti formátů .wav a .aiff
        if (reader != nullptr)
        {
            // načtení obsahu ze souboru
            auto newSource std::make_unique
            <juce::AudioFormatReaderSource>(reader, true);

            // nastavení obsahu objektu AudioTransportSource
            transportSource.setSource(newSource.get(), 0,
                                      nullptr, reader->sampleRate);

            // nastavení změny stavu
            stateChanged(Playing);

            //nastavení povahy tlačítka playButton
            playButton.setEnabled(true);

            // nastavení obsahu pro vykreslení zvukového průběhu
            thumbnail.setSource(new juce::FileInputSource(file));

            // předání obsahu pro další zpracování
            readerSource.reset(newSource.release());
        }
    }
}
```

A.2 Nastavení dialogového okna a volání funkce prepareForPlaying()

```
/**********************************************************************
*     Funkce browseButtonClicked() nastavující chování dialogového okna
*     a volající funkci prepareForPlaying()
*/

void BrowsefileAudioProcessorEditor::browseButtonClicked()
{
    // vytvoření objektu pro vyvolání dialogu a jeho nastavení
    chooser = std::make_unique<juce::FileChooser>
    ("Select a Wave file to play...", juce::File{}, "*.wav;*.aiff");

    // nastavení povahy pro object chooser
    auto chooserFlags = juce::FileBrowserComponent::openMode |
    juce::FileBrowserComponent::canSelectFiles;

    // vyvolání samotného dialogu
    chooser->launchAsync(chooserFlags,
                         [this](const juce::FileChooser& fileChooser)
    {
        // vytvoření souboru na základě výběru uživatele
        auto file = fileChooser.getResult();

        // předání vytvořeného souboru funkci prepareForPlaying()
        prepareForPlaying(file);
    });
}
```

A.3 Změna stavu programu na základě změny enumerátoru

```
/*************************************************************************
*     Funkce rozhoduje, která větev příkazu switch bude provedena na
*     základě změny stavu
*/
void BrowsefileAudioProcessorEditor::stateChanged(
      TranssportState newState)
{
    // příkaz testuje, zda došlo ke změně stavu
    if (newState != state) {
        state = newState;

        // příkaz rozhoduje, která větev bude zvolena na základě stavu
        switch (state)
        {
            case BrowsefileAudioProcessorEditor::Starting:
                playButton.setEnabled(false);
                stopButton.setEnabled(true);
                transportSource.start();              // přehrání zvuku
                break;
            case BrowsefileAudioProcessorEditor::Stoping:
                playButton.setEnabled(true);
                stopButton.setEnabled(false);
                transportSource.stop();              // zastavení zvuku
                break;
            case BrowsefileAudioProcessorEditor::Stoped:
                playButton.setEnabled(true);
                transportSource.setPosition(0.0);   // nastavení pozice
                break;
            case BrowsefileAudioProcessorEditor::Playing:
                playButton.setEnabled(true);
                stopButton.setEnabled(false);
                break;
            default:
                break;
        }
    }
}
```

A.4 Metody třídy DragAndDropTarget

```
/*************************************************************************
*     Funkce kontrolující formát souboru a vracející příslušnou hodnotu
*/
bool BrowsefileAudioProcessorEditor::isInterestedInFileDrag(
const juce::StringArray& files)
{
    // cyklus přes všechny soubory v poli files
    for (auto file : files)
    {
        // podmínka pro správnost souboru
        if (file.contains(".wav") || file.contains(".aiff")) {

            //vrácení příslušné návratové hodnoty
            return true;
        }
    }
    return false;
}

/*************************************************************************
*     Funkce, která na základě validnosti souboru nahraje daný soubor
*/
void BrowsefileAudioProcessorEditor::filesDropped(
const juce::StringArray& files, int x, int y)
{
    // cyklus přes všechny soubory v poli files
    for (auto file : files)
    {
        //testování, zda je soubor validní
        if (isInterestedInFileDrag(file))
        {
            // pokud ano, je nahrán pomocí funkce z přílohy A.1
            loadFile(file);
        }
    }

}
```

A.5 Kód pro načtení souboru do vyrovnávací paměti

```
/*****************************************************************
* Funkce nahrávající zvuková data do vyrovnávací paměti typu
* AudioSampleBuffer
*/

void BufferUnderstandAudioProcessorEditor::openButtonClicked()
{
    // předchozí kód odpovídá činnosti funkce v příloze A.2
    […]
    // testování validnosti načteného souboru
    if (file == juce::File{})
        return;

    // vytvoření čítače pro přečtení dat
    std::unique_ptr<juce::AudioFormatReader>
    reader(formatManager.createReaderFor(file));

    // pokud nastane správné předání informací čítači, provede se cyklus
    if (reader.get() != nullptr)
    {
        // zjištění doby trvání souboru
        auto duration =
        (float)reader->lengthInSamples / reader->sampleRate;

        // pro jednoduchost časové omezení délky souboru
        if (duration < 40)
        {
            // nastavení vyrovnávací paměti
            fileBuffer.setSize((int)reader->numChannels,
                               (int)reader->lengthInSamples);

            //přečtení zvukového souboru
            reader->read(&fileBuffer, 0, (int)reader->lengthInSamples,
                         0, true, true);
            position = 0;
            //nastavení pro přehrávání
            setAudioChannels(0, (int)reader->numChannels);
        }
    }
}
```

A.6 Kopírování obsahu vyrovnávací paměti do jiné vyrovnávací paměti

```
/*********************************************************************
* Funkce, která kopíruje část obsahu globální vyrovnávací paměti do
* jiné vyrovnávací paměti.
*/

void BufferUnderstandAudioProcessorEditor::playButtonClicked(
      juce::AudioSampleBuffer fileBuffer)
{
    // zastavení veškerého přehrávání (pro účely kontroly správnosi)
    // později s největší pravděpododobností odtraněno
    shutdownAudio();

    // nastavení velikosti vyrovnávací paměti
    buffer.setSize(fileBuffer.getNumChannels(), 44100);

    // cyklus pro překopírování dat ze všech kanálů souboru
    for (auto channel = 0; channel < fileBuffer.getNumChannels();
         channel++)
    {
        // příkaz pro kopírování obsahu
        buffer.copyFrom(channel, 0, fileBuffer, channel, 44100, 44100);
    }
    position = 0;
    // logická hodnota nastavuje přehrávání paměti s částí obsahu
    play = true;
    //nastavení reprodukce
    setAudioChannels(0, 2);
}
```

A.7 Objekt Channel

```
/*****************************************************************
*     Objekt Channel uchovávající ukazatel na počátek jednoduše
*     vázaného seznamu
*/

class Channel{
public:
    //identifikační řetězec
    juce::String ret;

    // ukazatel na začátek jednoduše vázaného seznamu granulí
    Grain* grainList;
};
```

A.8 Objekt Grain

```
/*****************************************************************
*     Kódové řešení objektu Grain
*/
class Grain {
public:
    //kontruktor pro vytvoření objektu
    Grain(int timeLength, int envelope, int density, int timeOffset, int
volume,  juce::AudioSampleBuffer  dataBuffer,  juce::AudioSampleBuffer
silenceBuffer) {
        this->delka = timeLength;
        this->obalka = envelope;
        this->pocet = density;
        this->offset = timeOffset;
        this->hlasitost = volume;
        this->grinOwnBuffer = dataBuffer;
        this->offsetBuffer = silenceBUffer;
        this->next = NULL;
    }
    //deklarace jednotlivých parametrů granule
    int delka;
    int obalka;
    int pocet;
    int offset;
    int hlasitost;
    juce::AudioSampleBuffer grinOwnBuffer;
    juce::AudioSampleBuffer offsetBuffer;
    Grain* next;

    //funkce pro odstraněníí granule z paměti
    void remove() {
        if (next != nullptr) {
            next->remove();
            delete next;
            next = nullptr;
        }
        delete this;
    }
};
```

Příloha B - Programové řešení projektů

Tato příloha je k práci přiložena v digitální podobě jako .zip archiv. Tento archiv obsahuje složku BP, kde se nacházejí soubory výsledného softwaru, složku SP, kde se nacházejí soubory vytvořené v rámci semestrální práce a složku VST3, kde se nachází výsledný plugin ve formátu .vst3.