VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2020 Štěpán Ježek

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

SYNTÉZA HUDEBNÍHO SIGNÁLU POMOCÍ PŘÍMÉHO GENEROVÁNÍ VYŠŠÍCH HARMONICKÝCH SLOŽEK
SYNTHESIS OF THE MUSICAL AUDIO SIGNAL USING DIRECT GENERATION OF HARMONICS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Štěpán Ježek

VEDOUCÍ PRÁCE
SUPERVISOR
RNDr. Lubor Přikryl

BRNO 2020

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: Štěpán Ježek ***ID:*** 203730
Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Syntéza hudebního signálu pomocí přímého generování vyšších harmonických složek

POKYNY PRO VYPRACOVÁNÍ:

Prozkoumejte syntézu hudebního signálu pomocí přímého generování vyšších harmonických složek a vytvořte zásuvný (plug-in) modul technologie VST s generováním jednoho tónu s možností editace hlasitosti všech vyšších harmonických složek. V rámci tohoto modulu vytvořte předvolby pro speciální případy (jen sudé harmonické, jen liché harmonické, a další), aplikujte na daný tón několik typů parametrizovatelných obálek tak, aby jednotlivé harmonické složky mohly mít různé parametry obálek, implementujte posun fundamentu a všech vyšších harmonických v temperovaném ladění tak, že každá vyšší harmonická bude mít hlasitost funkčně závislou na výšce fundamentu, a zahrňte parametry obálky do závislosti na frekvenci fundamentu. Implementujte řízení tohoto posunu přes MIDI.

DOPORUČENÁ LITERATURA:

[1] PIRKLE, Will. Designing Audio Effect Plug-Ins in C++. Focal Press, 2013. ISBN 978-0-240-82515-1.

[2] KIENTZLE, Tim. A Programmer's Guide to Sound. Addison-Wesley, 1998. ISBN 0-201-41972-6.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 8.6.2020

Vedoucí práce: RNDr. Lubor Přikryl

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

ABSTRAKT

Tato práce se věnuje problematice syntézy hudebního signálu, konkrétně tzv. aditivní (součtové) metodě. Hlavním cílem je implementace softwarového hudebního nástroje ve formátu zásuvného modulu VST3 v C++ aplikačním frameworku JUCE. Výsledný program umožňuje editaci harmonických složek modulového kmitočtového spektra a vytvoření časové závislosti složek pomocí tzv. morphingu mezi více, uživatelem nastavenými, stavy spektra. Úvod práce shrnuje obecně nejrozšířenější metody syntézy zvuku a porovnává je s metodou součtové syntézy. Následně je pojednáno o možných přístupech k implementaci tohoto typu syntézy, jejich přednostech, případně nedostatcích. Další sekce se zabývají technologiemi využitými k realizaci zásuvného modulu VST3 a rozborem hlavních částí tvořících výsledný program. Tento rozbor se zaměřuje především na část určenou pro zpracování signálu, uveden je ale také stručný popis jednotek tvořících grafické rozhraní.

KLÍČOVÁ SLOVA

Aditivní syntéza, C++, JUCE, syntezátor, virtuální nástroj, VST3, zásuvný modul

ABSTRACT

This thesis is focused on musical sound synthesis, in particular, the method of additive synthesis. The main goal is to implement a software musical instrument in the VST3 plug-in format, using the C++ programming language and the JUCE application framework. The final program offers spectral components editing capabilities and is able to morph between user-defined spectrum states in time. The introduction summarizes some common synthesis methods and their advantages or disadvantages. Next section deals with the technology used during the VST3 plug-in implementation and describes core parts that make up the final application. This analysis is focused mainly on the signal processing part, but there is also a brief description of the graphical user interface.

KEYWORDS

Additive synthesis, C++, JUCE, synthesiser, virtual instrument, VST3, plug-in

JEŽEK, Štěpán. *Syntéza hudebního signálu pomocí přímého generování vyšších harmonických složek*. Brno, 2020, 68 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: RNDr. Lubor Přikryl

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Syntéza hudebního signálu pomocí přímého generování vyšších harmonických složek“ jsem vypracoval samostatně pod vedením vedoucího bakalářské 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 bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu RNDr. Luboru Přikrylovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam výpisů

Úvod

Pro tvorbu umělého zvuku bylo historicky vyvinuto množství přístupů, jednou z nejstarších a nejvýznamnějších metod je tzv. součtová (někdy též aditivní) syntéza, která výsledný zvuk popisuje jako průběh aproximovaný součtem jednodušších signálů. Teoreticky lze dle Fourierovy analýzy tímto, relativně intuitivním způsobem, vytvořit libovolný hudební signál.

Větší rozšíření této metody však bylo v minulosti, především pro výpočetní nedostatky technických prostředků, značně omezeno. Obvykle se jednalo o komerčně nedostupná zařízení, součtové syntezátory se tedy většinou využívaly v rámci výzkumů nebo výraznějších hudebních projektů. Případně bylo nutně pro implementaci vytvořit jednodušší technické řešení, které mělo do určité míry negativní vliv na kvalitu zvuku.

V současné době již pro realizaci existují dostupné technické prostředky a vzniká řada syntezátorů, které implementují součtovou metodu buď přímo jako hlavní způsob syntézy nebo alespoň jako jednu z nabízených variant pro generování zvuku. Součtová syntéza je tedy aktuálně dostupnější a důsledkem toho i více rozšířená. Stále patří mezi klasické přístupy k tvorbě umělého zvuku a nalézá využití také jako prostředek k analýze či zkoumání signálů.

Tato práce si klade za cíl realizaci softwarového nástroje založeného na principech a poznatcích součtové syntézy s důrazem na možnost editace a obecnější funkční závislosti harmonických složek.

Úvodní kapitola se zabývá základními vlastnostmi zvukového signálu s přihlédnutím k jeho charakteristice z hlediska hudby a souvislostmi tohoto charakteru s časovým a frekvenčním průběhem.

Následně bude uvedeno základní rozdělení metod tradičně využívaných pro zvukovou syntézu, čímž bude součtová syntéza zasazena do širšího kontextu.

V navazující kapitole jsou shrnuty některé z možných přístupů k implementaci aditivního syntezátoru, jsou vyhodnoceny přednosti a nedostatky jednotlivých metod z hlediska zvukových možností a výpočetní náročnosti.

Následující kapitoly se věnují tvorbě syntezátoru v jazyce C++. Shrnují technologie JUCE framework a VST (hlavní nástroje použité pro realizaci) a specifikaci parametrů výsledné aplikace. Následně pojednávají o konkrétních softwarových strukturách, které zajišťují např. generování signálu, časovou závislost harmonických složek či polyfonii.

1 Analýza hudebního signálu

Signálem se obecně rozumí přenos informace časově proměnnou fyzikální veličinou. V případě zvukového signálu dochází, prostřednictvím např. změn tlaku v plynném prostředí, k přenosu zvukové informace. Pokud je zvukový signál nositelem hudební informace, hovoříme o tzv. hudebním signálu. K popisu tohoto typu signálů se využívají standardní veličiny jako jsou frekvence, amplituda a fáze, v souvislosti se zvukem označované jako tzv. objektivní vlastnosti. U zvuku je důležitý také sluchový vjem, který se popisuje pomocí tzv. subjektivních vlastností.

Základní subjektivní vlastnosti zvukového signálu charakterizují jeho výšku, hlasitost a barvu. U všech těchto vlastností lze hledat určitou spojitost s vlastnostmi objektivními. V základním zjednodušení a při uvažování zvukového signálu tónového charakteru je vjem výšky tónu závislý na frekvenci a vjem hlasitosti odráží amplitudu signálu. Vnímání zvukové barvy obecně souvisí s časovým průběhem a frekvenční strukturou signálu, její přesnější určení a hledání souvislostí s objektivními vlastnostmi je v porovnání s výškou a hlasitostí poměrně komplikované.

Pro objektivní vlastnosti signálu platí, že jsou na sobě nezávislé, změna frekvence tedy např. neovlivní amplitudu. Subjektivní vlastnosti naopak závislé jsou a změny např. výšky mohou způsobit také změnu barvy nebo hlasitosti.

Hudební signál je svázán s časem jako nezávisle proměnnou veličinou, signál samotný je tak typicky vymezen reálnou délkou trvání a jeho vlastnosti jsou zpravidla závislé na čase. Jako analytický prostředek se využívá i tzv. statické pojetí hudebního signálu, které uvažuje signál s nekonečnou délkou a neměnnost všech jeho vlastností. Při určitých zjednodušeních se s využitím tohoto pojetí např. definuje tón a hluk jako signál periodický a s danou výškou (tón) nebo naopak neperiodický, u kterého výška nelze určit (hluk). Prakticky však toto pojetí nelze uplatnit všeobecně, např. výšku lze z hlediska psychoakustiky subjektivně přiřadit i hluku.

Přístup, který zohledňuje konečnou dobu trvání a časové závislosti se označuje jako tzv. dynamické pojetí hudebního signálu. [1, 2]

1.1 Časové vlastnosti

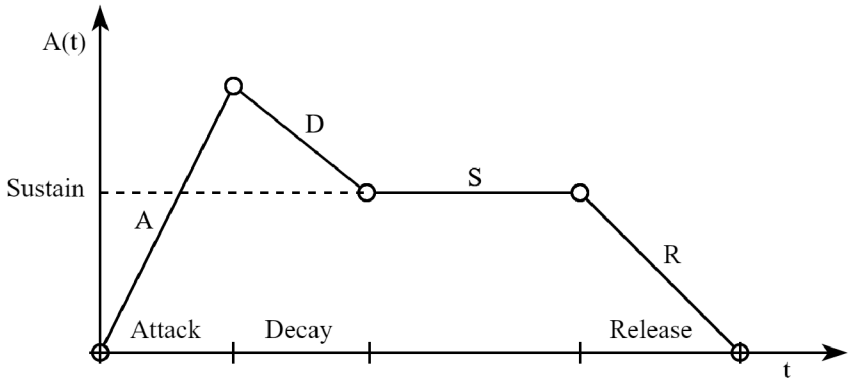
Mezi základní abstrakci hudebního signálu patří jeho časový průběh. Změny v časovém průběhu od vzniku signálu po doznění mohou pomoci alespoň přibližně určit jeho základní charakteristiky. Opakující se úseky jsou typické pro signál tónového charakteru, náhodné změny naopak značí hluk nebo šum. Dále je možné zhruba určit také frekvenční strukturu signálu, rychle proměnlivá amplituda značí obsah vyšších frekvenčních složek, naopak jednodušší časový průběh je příznačný pro nižší podíl vyšších frekvencí.

Z časového průběhu je možné dále odvodit tzv. časovou obálku signálu, která se využívá pro další kategorizaci hudebních signálů, např. k rozlišení perkusního nebo neperkusního charakteru hudebního nástroje. Obecně je možné obálku vyjádřit jako modulační funkci, která upravuje amplitudu signálu, výsledný signál $s(t)$ je tedy dán vztahem

$$s(t) = A(t) \cdot g(t) \tag{1.1}$$

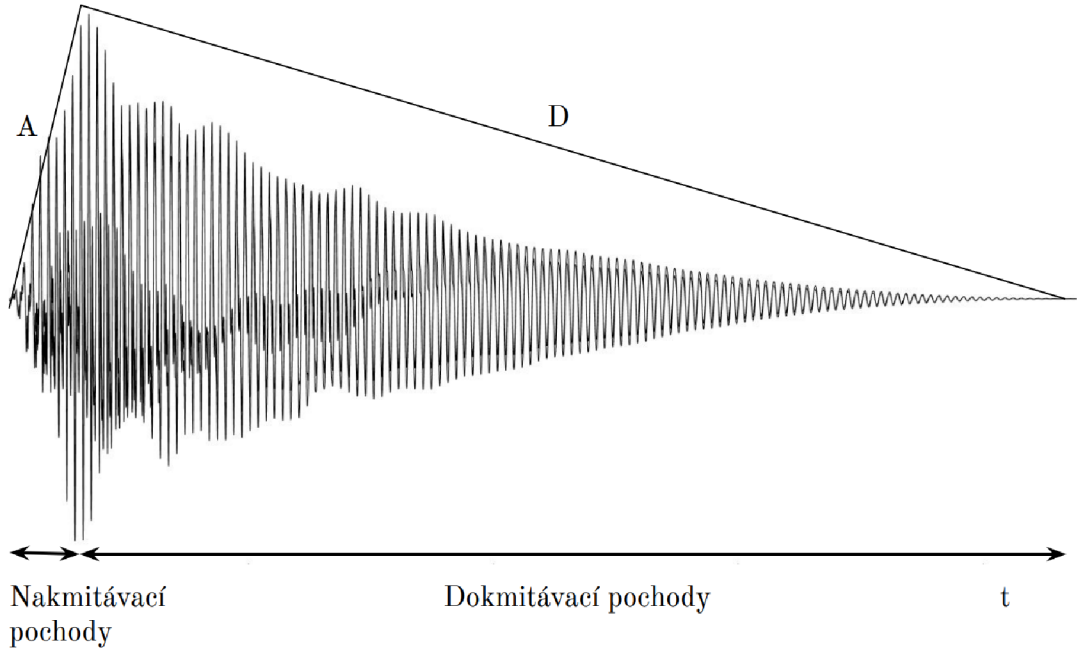
kde $A(t)$ modulační funkce a $g(t)$ modulovaný signál. V obálce se rozlišují časové oblasti tzv. *nakmitávacích pochodů* (někdy označované jako *tranzient*), *zakmitaný stav* a oblast *dokmitávacích pochodů*. Přechody mezi jednotlivými stavy nejsou většinou u reálných nástrojů jednoduše rozpoznatelné, v průběhu zakmitaného stavu však platí, že dochází k malým změnám (frekvence, amplitudy a fáze) malou rychlostí, nakmitávací a dokmitávací pochody vykazují velké změny velkou rychlostí.

Zvuková syntéza (viz kapitola 2) tento průběh nejčastěji modeluje pomocí tzv. *ADSR* obálky (viz obr. 1.1), kterou tvoří fáze A – attack, D – decay, S – sustain a R – release. Fáze attack určuje dobu, za kterou generovaný signál dosáhne maximální hodnoty, fáze decay čas poklesu na hodnotu zakmitaného stavu, sustain úroveň signálu v zakmitaném stavu a release dobu dokmitávacích pochodů.

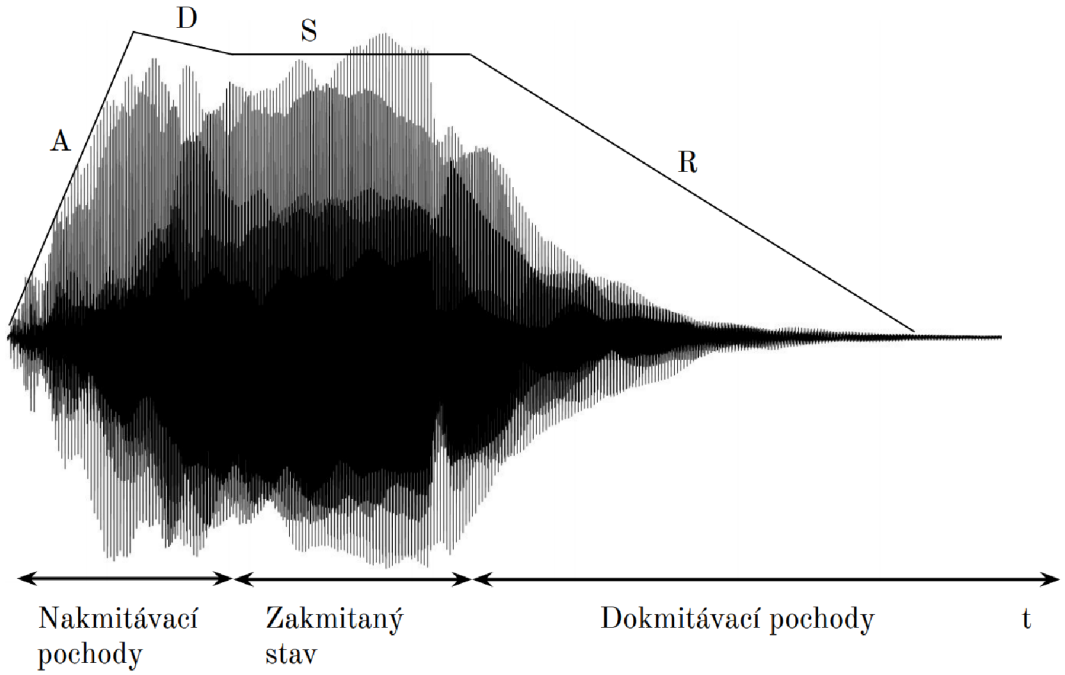


Obr. 1.1: Průběh obálky typu ADSR

Pro perkusní zvuky je typické, že zakmitaný stav nevykazují, po nakmitávacích pochodech bezprostředně nastupuje výraznější fáze dokmitávacích pochodů (viz obr. 1.2). U zvuků neperkusních naopak lze v určité míře zakmitaný stav identifikovat (viz obr. 1.3). Rozdělení perkusních a neperkusních zvuků souvisí se způsobem, kterým je kmitajícímu elementu daného zvukového zdroje dodávána energie. Pokud se jedná o impulzní buzení (např. úder kladívka), hovoříme o perkusním zvuku, v případě souvislého předávání energie (např. tah smyčce) je výsledný zvuk neperkusní. [1]



Obr. 1.2: Perkusní signál a jeho časová obálka (housle – pizzicato)



Obr. 1.3: Neperkusní signál a jeho časová obálka (trubka)

1.2 Frekvenční vlastnosti

Uspořádání frekvenčních složek signálu se označuje jako tzv. *frekvenční struktura*, její grafická nebo numerická reprezentace jako tzv. *frekvenční spektrum*. Charakter spektra je jedním ze klíčových parametrů pro klasifikaci hudebních signálů.

Základní dělení rozlišuje *stacionární* a *nestacionární* signály. Vlastnosti stacionárních signálů se v dostatečně dlouhém časovém intervalu nemění, pro jejich analýzu tedy není rozhodující volba počátku časové osy. Nestacionární signály vykazují velkou variabilitu těchto vlastností v čase a závisí proto na zvoleném počátku časové osy.

Stacionární signály se dále rozdělují na *deterministické*, které jsou pro každý časový okamžik přesně definovány matematickým vztahem, nebo *stochastické* – náhodné, pro jejich popis se využívají metody pravděpodobnosti a statistiky. V hudebním pojetí mají stochastické signály obvykle šumový nebo hlukový charakter a jejich spektrum je spojité. Pokud je deterministický signál tvořen složkami, jejichž frekvence jsou celočíselnými násobky frekvence základní, označuje se jako *periodický*. Spektrum periodického signálu je diskrétní (čárové), jednotlivé spektrální čáry reprezentují harmonické složky. V případě, že frekvence složek nejsou v přesném celočíselném poměru, signál je tzv. *kvaziperiodický*.

V reálných podmínkách hudební signály vykazují vlastnosti více uvedených typů současně, pouze jeden je však obvykle převažující. U delších vydržovaných tónu dominuje periodický typ, některé laděné perkusní nástroje (např. zvony) mají převažující spíše kvaziperiodický charakter, stochastický signál představuje např. činel. [1]

2 Syntéza hudebního signálu

Syntézou se obecně označuje sjednocení, sestavení či skládání se z více částí, v hudebním kontextu se s tímto výrazem lze nejčastěji setkat v souvislosti s tvorbou umělého zvukového signálu. Nový zvuk může vzniknout úpravou již existujícího materiálu nebo elektronickým či mechanickým generováním. Záměrem syntézy může být co nejvěrnější imitace přirozených zvukových vjemů (např. tradičních akustických hudebních nástrojů) nebo vytváření nových zvukových barev a většinou bývá spojována s elektronickými hudebními nástroji – syntezátory.

K realizaci syntézy existuje řada metod, nejvýznamnější z nich shrnují následující podkapitoly.

V nejzákladnějším pojetí všechny metody využívají kombinaci signálového generátoru (oscilátoru) a následně sekce pro úpravu tohoto signálu, způsob generace a zpracování je však specifický pro každou metodu. Podle typu elektronických obvodů použitých při implementaci dané metody je možné rozlišit syntézu analogovou, hybridní (zpravidla tvořena analogovým generátorem s digitálním řízením) a digitální. Tato práce se zaměřuje na principy digitální syntézy. [3]

2.1 Součtová (aditivní) syntéza

Metoda součtové syntézy je založena na vytváření komplexních signálů součtem určitého počtu signálů jednodušších. V klasickém případě vychází z tzv. Fourierovského přístupu, který jako jednodušší složky využívá signály harmonického průběhu. Pokud je amplituda, frekvence a fáze složek konstantní, hovoříme o tzv. statické syntéze. Při dostatečném počtu harmonických je možné dosáhnout poměrně velké zvukové variability, časová neměnnost však může vytvářet strohý, chladný zvuk.

Oproti tomu tzv. dynamický model součtové syntézy umožňuje nezávislé řízení amplitudy, frekvence i fáze jednotlivých složek. Implementace tohoto modelu může obsahovat velké množství nastavitelných parametrů, obvykle se tedy využívají tzv. makropovely, které mohou navázat více parametrů k jednomu ovládacímu prvku.

Nefouriefovský přístup součtové syntézy skládá výsledný signál z komplexních složek, neomezuje se pouze na sinusové signály. Pomocí relativně malého počtu složek, tedy i s nižší technickou náročností lze získat spektrálně bohatý signál.

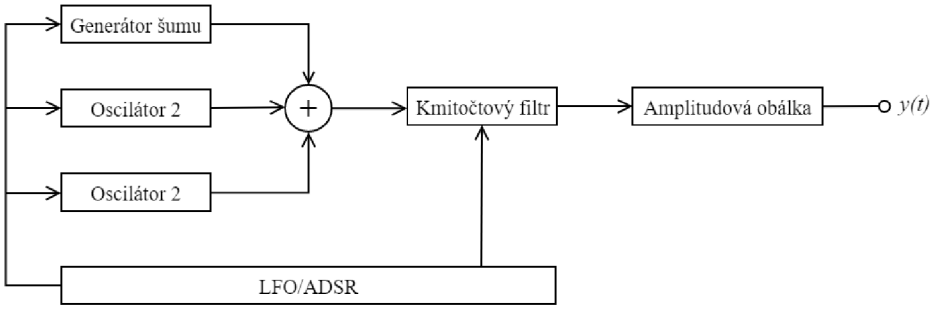
Součtová syntéza bývá považována, především pro možnost přesné kontroly generačního procesu, za nejdokonalejší způsob tvorby umělého zvuku. Hlavní nedostatek této metody představuje technicky náročná implementace, která má typicky pro velký počet generátorů a obálek vyšší nároky na výpočetní výkon. V praxi tedy syntezátory využívají řadu zjednodušení, více harmonických složek může například

sdílet společnou amplitudovou obálku (metoda tzv. parciálních témbrů), případně lze použít jednu obálku pro všechny složky. [1, 4]

2.2 Rozdílová (subtraktivní) syntéza

Tato metoda využívá komplexní signál s harmonicky bohatým spektrem, který je následně zpracován filtrem s řízenou mezní frekvencí. Nejčastěji se používají signály pilového, obdélníkového a trojúhelníkového průběhu v kombinaci s bílým nebo růžovým šumem.

Hlavní součástí metody je filtr typu dolní, horní nebo pásmová propust, u kterého lze upravovat frekvenci, obvykle také činitel kvality (v syntezátorech často označovaný jako resonance) a strmost. Realizace rozdílové syntézy zpravidla dále obsahují amplitudovou obálku a nízkofrekvenční oscilátor (*LFO*), kterým je možné modulovat vybrané parametry (např. frekvenci oscilátoru nebo filtru).



Obr. 2.1: Klasická struktura rozdílového syntezátoru

Pro jednodušší technickou realizaci a současně dobré zvukové možnosti se rozdílová syntéza prosadila s prvními komerčně úspěšnými syntezátory a dodnes patří mezi nejpoužívanější metody. [1]

2.3 Modulační syntéza

Metody modulační syntézy jsou založeny na řízených změnách parametrů signálu (nosné) signálem modulačním. Ve zvukové syntéze se nejčastěji používá frekvenční modulace *(FM)*, která je realizována soustavou jednoduchých sinusových oscilátorů (operátorů) spojených do zvoleného schématu, kdy jeden oscilátor řídí frekvenci dalšího. Změnou uspořádání je možné jednotlivé operátory využívat jako generátor nosné vlny nebo jako modulátory.

Hlavní předností *FM* syntézy jsou rozsáhlé zvukové možnosti. Poměrně jednoduchým způsobem lze generovat komplexní průběhy tónového nebo šumového charakteru (typicky využívané pro různé ruchy či perkuse). Výsledné spektrum generovaného signálu je možné matematicky popsat pomocí Besselových funkcí, přesto v praxi často zvukový výsledek bývá značně nepředvídatelný a cílené programování konkrétních zvuků vyžaduje hlubší znalosti syntézy.

Mezi další modulační metody se řadí, nepříliš rozšířená, amplitudová modulace a kruhová modulace, která se obvykle nepoužívá jako autonomní prostředek syntézy, spíše jako volitelná složka v syntezátorech založených na rozdílové metodě. [4]

2.4 Tvarová syntéza

Oproti předchozím metodám, ve kterých je kladen důraz na frekvenční spektrum vytvářeného signálu (součet harmonických složek, filtrace nebo vytváření nových spektrálních složek pomocí modulace) se tvarová syntéza primárně soustředí na úpravu generovaného signálu v časové oblasti.

Manipulace časového průběhu může být dosaženo například nelineárním tvarováním sinusového signálu systémem s nelineární přenosovou charakteristikou, přímým zadáním časového průběhu, aproximací jednoduchého (zpravidla sinusového) signálu jednoduššími parabolickými oblouky nebo segmentací, při které se výsledný průběh skládá ze segmentů zadaných křivek (např. hyperboly, paraboly či exponenciály).

K tvarový metodám patří také, v současnosti relativně populární, granulační syntéza, která signál vytváří přehráváním krátkých úseků získaných z existujícího zvukového materiálu. Dále tvarovou metodu využívá také tzv. lineárně-tvarová syntéza, někdy označovaná jako sampling. [1]

2.5 Modelující syntéza

Metoda modelující syntézy (též matematické modelování) se soustředí v první řadě na principy a funkce modelovaného nástroje, snaží se napodobit procesy, které při vzniku zvuku v nástroji probíhají. Neuvažuje se, oproti předchozím metodám, pouze zvukový výsledek, ale i jednotlivé fáze, ve kterých vzniká a mění se.

Realizace tohoto přístupu zpravidla obnáší vytvoření komplexního matematického modelu popisujícího daný fyzikální systém nebo jeho části (např. kmitající struny nebo vzduchový sloupec), který na základě poskytnutých vstupních parametrů provede výpočet sady matematických vztahů. Z toho důvodu se modelující metody vyznačovaly vyšší výpočetní náročností, v současné době jsou však již dostupné

komerční softwarové produkty modelující řadu tradičních akustických nástrojů při relativně nízkých požadavcích na výkon. [1, 4]

3 Metody součtové syntézy

Součtová syntéza patří mezi první metody, které byly využívány při tvorbě hudby s pomocí počítačů. V 60. letech 20. století byly součtovou metodou vytvořeny jedny z prvních umělých zvuků, které věrně napodobovaly tradiční akustické hudební nástroje. V následujícím období byla v rámci výzkumů věnujících se obecně tzv. spektrálnímu modelování[1] vytvořena řada přístupů k implementaci součtové syntézy. [5]

3.1 Přímé generování harmonických složek

Historicky nejstarší způsob realizace součtové syntézy využívá soustavu harmonických oscilátorů a součet jejich výstupních signálů v nastaveném poměru. Jedná se tzv. statický model součtové syntézy, který je možné popsat vztahem

$$s(t) = \sum_{k=1}^{N} A_k \sin(\omega_k t + \phi_k) \tag{3.1}$$

kde A_k je amplituda, ω_k je frekvence a ϕ_k počáteční fáze k-té harmonické složky z celkového počtu N složek.

Jednotlivé harmonické složky lze modulovat soustavou amplitudových, případně také frekvenčních obálek. V takové případě se jedná o tzv. dynamický model, který je popsán vztahem

$$s(t) = \sum_{k=1}^{N} A_k(t) \sin[\omega_k(t) \cdot t + \phi_k(t)] \tag{3.2}$$

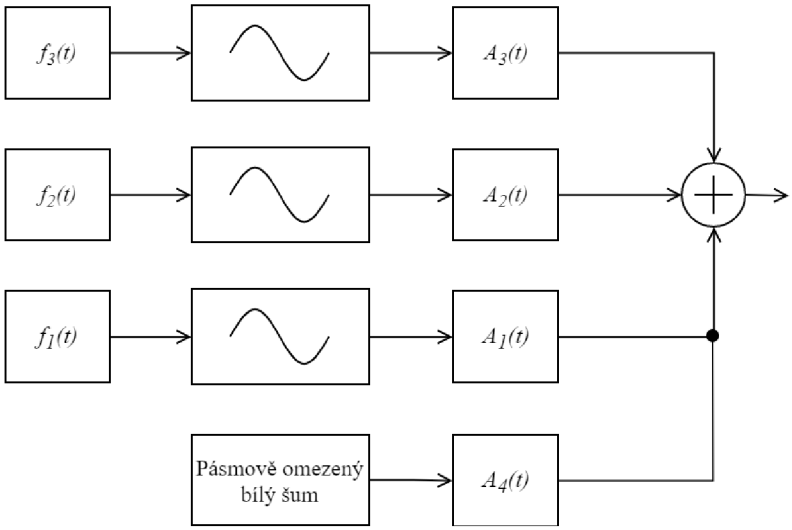
kde $A_k(t)$, $\omega_k(t)$ a $\phi_k(t)$ udává časovou závislost amplitudy, frekvence a fáze k-té harmonické složky. [1]

V hudebních signálech je obecně významná také přítomnost šumu, který je možné modelovat harmonickými signály s frekvencemi blízkými natolik, že je lidským sluchem nelze vnímat jednotlivě. Pro nutnost velkého počtu harmonických složek je však efektivnější využít generátor bílého šumu a kmitočtový filtr (viz Obr.3.1). Časová závislost spektra takového modelu je tvořena deterministickou (harmonické generátory) a stochastickou složkou (bílý šum filtrovaný podle funkce času)

$$s(t) = \sum_{k=1}^{N} A_k(t) \sin[\omega_k(t) \cdot t + \phi_k(t)] + e(t) \tag{3.3}$$

[1] Oblast spektrálního modelování shrnuje metody zabývající tvorbou hudebních i nehudebních signálů vytvořením odpovídajícího frekvenčního spektra, případně dalšími úpravami signálu ve frekvenční oblasti. Mimo součtovou syntézu lze do těchto metod zahrnout např. syntézu s využitím *frekvenční modulace* (FM) nebo vokodéry.

kde $A_k(t)$, $\omega_k(t)$ a $\phi_k(t)$ udává časovou závislost amplitudy, frekvence a fáze k-té harmonické složky, $e(t)$ označuje bílý šum po zpracování kmitočtovým filtrem.



Obr. 3.1: Součtová syntéza využívající model tón + šum

Využití filtrovaného bílého šumu představuje efektivní kombinaci rozdílové a součtové syntézy.

Poměry harmonických složek a šumu, případně jejich časové závislosti mohou být zadány manuálně uživatelem, pokud je však cílem přesná reprodukce daného signálu, využívá se automatická analýza, která samostatně vytvoří časové závislosti amplitudy a frekvence pro každou harmonickou složku. V případě, že vstupní signál obsahuje tranzienty nebo různé rychlé časové změny, je nutné zohlednit také fázi složek. U časově stálých tónových průběhů není fáze lidským sluchem vnímána a proto je možné ji u syntézy zanedbat.

Při analýze se vstupní signál v čase rozdělí na krátké úseky, které se pomocí FFT převedou do frekvenční oblasti. V těchto úsecích se následně detekují časové změny amplitudy a frekvence každé harmonické složky. Výsledkem analýzy je sada amplitudových a frekvenčních (příp. také fázových) obálek, pomocí kterých se daný průběh s využitím interpolace (obvykle lineární) reprodukuje soustavou harmonických oscilátorů. [5]

3.2 Inverzní FFT

Pro snížení výpočetních nároků většího počtu harmonických oscilátoru byla vytvořena metoda, která požadovaný signál vytváří zpětnou Fourierovou transformací odpovídajícího frekvenčního spektra. Výpočetní náročnost této metody nezávisí na

počtu složek, dále lze do signálu ve frekvenční oblasti relativně jednoduše přidat složky neharmonického průběhu, např. pásmově omezený šum.

První metoda využívající IFFT byla navržena v 80. letech 20. století, spektrum signálu se vytvořilo prostou manipulací s hodnotami vzorků ve frekvenční oblasti tak, aby bylo dosaženo požadované amplitudy a fáze jednotlivých složek po následném převodu do časové oblasti.

Další navržená metoda předchozí přístup rozšířila o možnost změny amplitudy harmonických v čase. Oproti přímému nastavení hodnot jednotlivých vzorků ve frekvenční oblasti se harmonická složka do spektra přidá vložením hlavního laloku okénkové funkce posunutého na pozici odpovídající frekvenci dané složky. Příspěvek složky ve spektru $S_l[k]$ je určena vztahem

$$A_k \cdot e^{i\phi k} \cdot W[f_k - k], \tag{3.4}$$

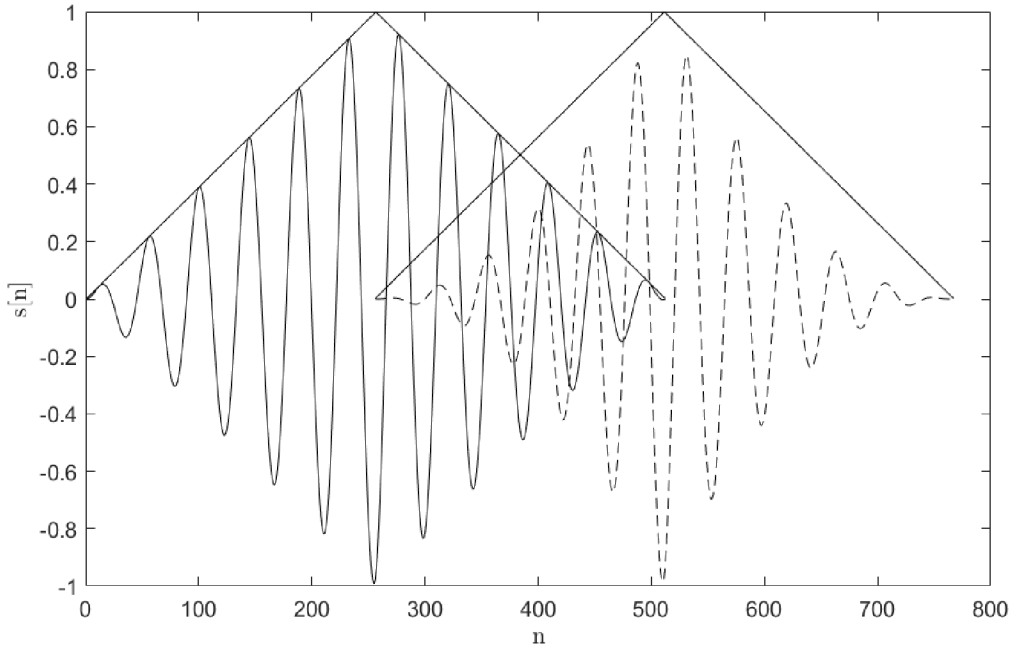
kde A_k, f_k a ϕ_k jsou střední hodnoty průběhů amplitudy, frekvence a fáze $k-$té harmonické složky v krátkodobém komplexním spektru S_l a W je Fourierova transformace okenní funkce w, která je posunuta tak, aby byla symetrická kolem f_k.

Postranní laloky okénkové funkce se zanedbají. Po převedení do časové oblasti je amplituda daného úseku tvarována podle okna použitého ve frekvenční oblasti. Pro celkový průběh požadovaného signálu se vytvoří sada tzv. krátkodobých spekter (*Short Term Spectrum* (STS)), které se po zpětné transformaci překryjí v čase a díky tvarování amplitudy okénkovou funkcí na sebe plynule navazují (viz Obr.3.2). Je nutné zvolit funkci okna s užším hlavním lalokem ve frekvenčním spektrum, aby bylo možné modelovat dostatečně úzké frekvenční pásmo. Dále je žádoucí, aby během překrývání jednotlivých bloků v časové oblasti byla amplituda konstantní. Pro splnění obou požadavků bylo jako vhodný kompromis vybráno trojúhelníkové okno.

Nedostatkem této metody syntézy je statická frekvence jednotlivých harmonických v krátkodobém spektru, při větší rychlosti frekvenčních změn (např. při modelování glissanda) mohou vzniknout neplynulé přeskoky frekvence mezi jednotlivými bloky. Dále metoda poskytuje nižší zvukovou kvalitu v porovnání s přímým generováním složek. [5, 6, 7]

3.3 Wave Table syntéza

Pokud je generovaný signál periodický, je možné výsledný průběh vytvářet pomocí známých hodnot jediné periody. Generování libovolného periodického signálu se tím zjednoduší na interpolaci mezi známými hodnotami, čímž lze výrazně snížit výpočetní náročnost. Jedná se obecně o optimalizační techniku, která se v oblasti



Obr. 3.2: Časový průběh signálu vytvořeného metodou využívající IFFT.

číslicového zpracování signálů (DSP) často využívá.

Na stejném principu je založena také Wave Table syntéza, jedna perioda vybraného průběhu (tzv. wavetable) je předem uložena do tabulky, s použitím interpolace (obvykle lineární, příp. kvadratické) je možné generovat signál s daným průběhem o libovolné frekvenci.

Pro realizaci součtové syntézy se tabulka naplní hodnotami jedné periody signálu složeného z vybraných harmonických složek. Vygenerovaný signál se zpravidla dále tvaruje amplitudovou obálkou, v tomto případě jsou však ovlivněny amplitudy všech harmonických. Měnit poměry jednotlivých složek v čase lze prolínáním (tzv. morphing) mezi tabulkami v čase. [5]

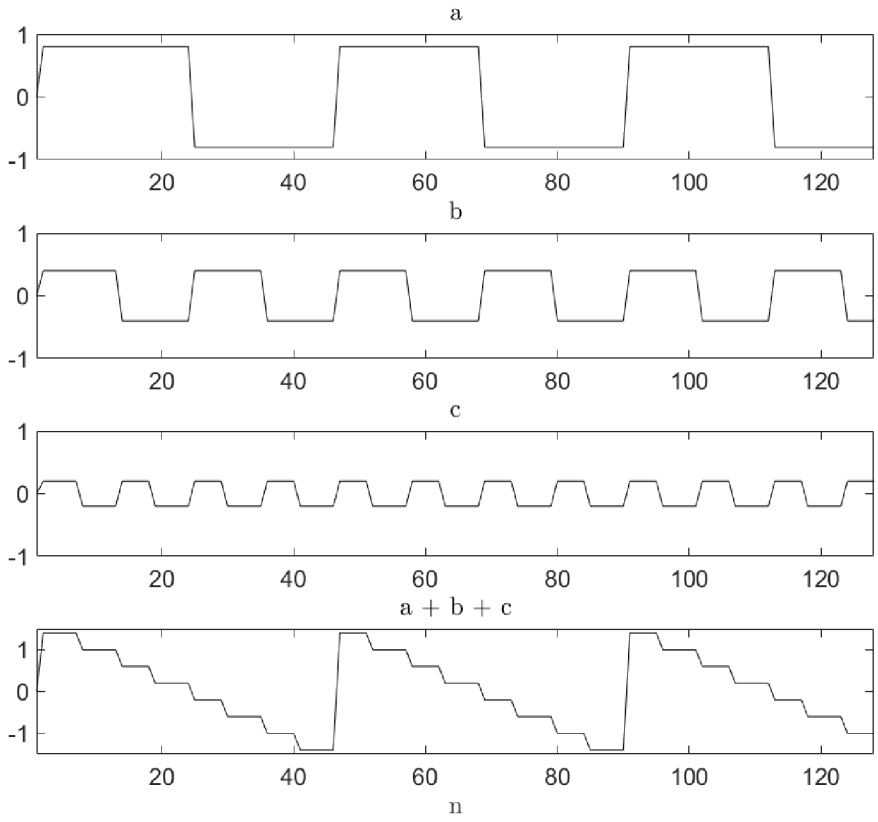
Nevýhodou metody je vznik aliasingu při generování vyšších frekvencí, který je nutné potlačit pomocí jiných technik (např. převzorkováním nebo frekvenčním omezením signálu v tabulce). I přesto se však jedná o velmi používaný způsob implementace oscilátorů na softwarových syntezátorech. [8]

3.4 Nefouriefovské součtové metody

Využití nefourierovských metod představuje další způsob jak snížit obecně vyšší technickou náročnost součtové syntézy. Nejznámější metoda využívá systém Walsho-

vých funkcí, které je založena aproximaci výsledného signálu součtem obdélníkových signálů (viz Obr.3.3), které lze oproti harmonickému průběhu generovat s menšími nároky.

V praxi však tyto metody, i přes četné technické výhody, našly využití pouze k výzkumným účelům, protože představa vzniku tónu tímto způsobem se v běžném hudebním kontextu neprosadila. [1]



Obr. 3.3: Aproximace pilového průběhu součtem obdélníkových signálů

4 Návrh aplikace

Tato kapitola pojednává o celkové specifikaci výsledného programu, shrnuje nástroje použité k implementaci a základní programové koncepty zajišťující hlavní funkce aplikace. Stručně se charakterizuje pojem zásuvných modulů v souvislosti s technologií *Virtual Studio Technology* (VST), která bude v této práci využívána v rámci aplikačního frameworku JUCE. Tomu je spolu s programem *Projucer*, zajišťujícím správu projektů a sestavení aplikací postavených na *JUCE*, věnována další podkapitola.

Následně bude uveden rozbor tříd, implementujících vysokoúrovňovou funkci syntezátoru jako obecně zařízení, které je schopné přijímat hudební informace a na jejich základě řídit generování signálu. *JUCE* framework tento model realizuje v rámci tří poskytnutých tříd, jejich stěžejní metody budou v podkapitole popsány.

Závěr kapitoly se zabývá metodami implementace harmonického oscilátoru a generátoru obálky.

Všechny použité knihovny, samotná aplikace i níže uvedené příklady jsou napsány v jazyce C++, který se, pro možnost vysoké rychlost běhu programu a nízkoúrovňových optimalizací, standardně využívá pro software zpracovávající zvuk.

4.1 Specifikace

Cílem práce je vytvoření zásuvného modulu syntezátoru, schopného pracovat v reálném čase a s grafickým rozhraním, skrze které by bylo možné intuitivně ovládat větší počet parametrů součtové syntézy. Primárním záměrem je využít aplikaci pro zvukovou produkci v rámci standardních programů typu *Digital Audio Workstation* (DAW) nebo jako prostředek pro analýzu případně ukázku procesu tvorby komplexního hudebního signálu z jednodušších složek.

Generování a úprava signálu

Hlavní součástí aplikace bude modul umožňující nastavení úrovní harmonických složek generovaného signálu. Všechny složky budou v reálném čase generovat harmonický signál s frekvencí, která bude celočíselným násobkem základní frekvence. Výpočet jednotlivých složek i s využitím některých optimalizačních metod (např. generování pomocí tabulky) představuje vyšší nároky na výpočetní výkon, výhodou však je možnost redukovat harmonické složky s vyšší frekvencí než $f_{vz}/2$, čímž lze relativně jednoduchým způsobem potlačit aliasing. Pro generování tónu se v závislosti na aktuální frekvenci signálu budou využívat pouze složky v pásmu, ve kterém nedochází k aliasingu. Základní frekvence se bude nastavovat podle čísla noty dle

standardu MIDI a v průběhu generování bude možné notu transponovat o uživatelem zadaný počet oktáv a půltónů.

Počet harmonických složek byl jako určitý kompromis mezi náročností výpočtu a zvukovými možnostmi nástroje stanoven na 64. Fáze složek je fixní, důvodem je především nižší význam této veličiny pro sluchový vjem ustáleného hudebního signálu, u kterého není fáze vnímána. Větší význam má pouze u rychlých změn průběhu, např. u tranzientů perkusních signálů [5].

Rychlé nastavení složek na některý ze standardních průběhů běžně využívaných ve zvukové syntéze bude možné pomocí předvoleb pro pilový, obdélníkový, trojúhelníkový a harmonický signál.

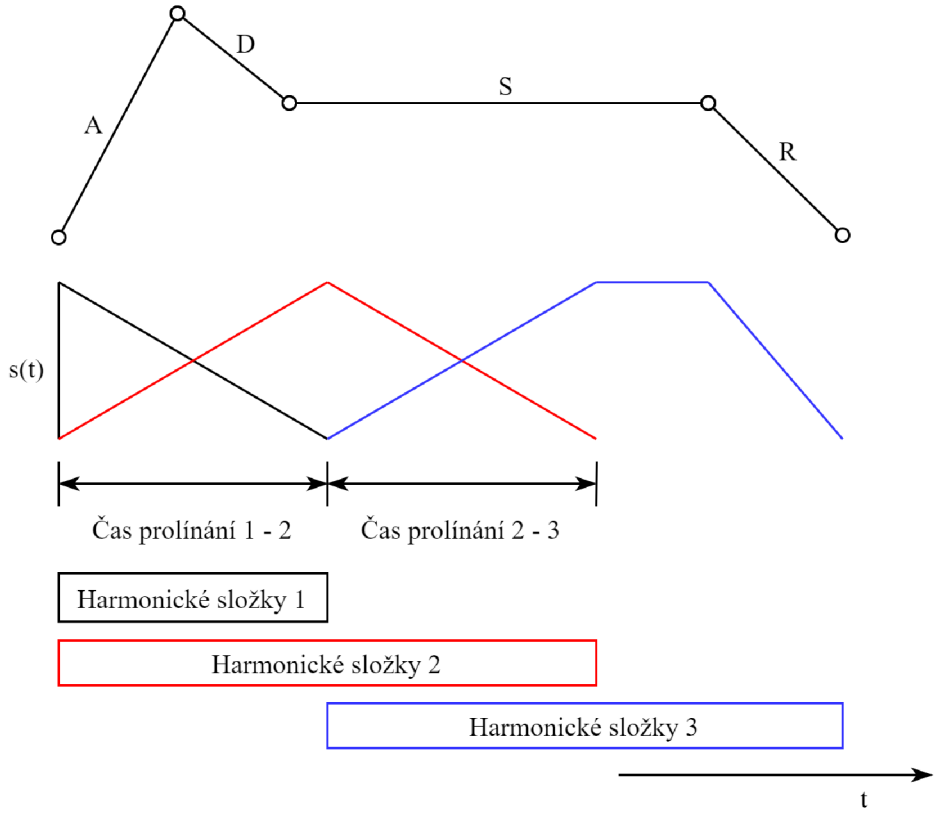
Pro modelování časové závislosti složek bude aplikace uživateli nabízet vytvoření až 8 vlastních poměrů harmonických, mezi kterými bude následně možné prolínat (tzv. *morphing*) s nastavenou dobou trvání. Průběh takto vygenerovaného signálu bude dále upraven ještě obálkou typu ADSR, která bude na časové změně harmonických fungovat nezávisle. Pokud se časová modulace dostane k poslednímu vytvořenému stavu a generovaný tón je stále držen (klávesa stisknuta), zůstanou harmonické v poměru posledního stavu a dále se nebudou měnit. Koncept modulace amplitudy složek souhrnně ilustruje Obr.4.1.

Okrajově lze ještě zmínit, že se tímto přístupem nepřímo realizuje také tzv. *vektorová syntéza*, která je založena na postupném prolínání různých průběhů signálů v časové oblasti.

Pro další úpravu signálu budou součástí také kmitočtové filtry typu dolní, horní a pásmová propust. Úpravy spektra generovaného signálu by teoreticky bylo možné dosáhnout manuální úpravou harmonických. Pomocí modulu filtru je však možné vytvořit modifikátor nezávislý na frekvenci fundamentu, čímž se dále rozšiřují možnosti syntézy a případně mohou zohlednit tzv. *formanty*, spektrální složky hudebního signálu, jejichž frekvence se nemění s výškou tónu. Filtr bude dále obsahovat také tzv. sledovač klaviatury, díky kterému bude možné docílit, aby pro každou zahranou notu byl zachován poměr frekvence této noty k mezní frekvenci filtru. Míru zachování tohoto poměru bude možné nastavit pomocí ovládacího prvku, přičemž maximální poloha prvku zajistí fixní poměr pro celý rozsah klaviatury, v minimální poloze nebude mezní frekvence filtru výškou generovaného tónu ovlivněna.

Posledními součástmi řetězce pro zpracování signálu bude dozvukový efekt (reverb) umožňující vytvořit dojem reálného poslechového prostoru a jednoduchý modul určující zesílení celkového výstupního signálu pro omezení zkreslení, které může vzniknou při hře více hlasů současně.

Syntezátor bude schopen generovat až 4 tóny nezávisle – bude disponovat čtyřhlasou polyfonií. Celkové uspořádání jednotlivých sekcí je uvedeno v blokovém schématu na Obr.4.2.



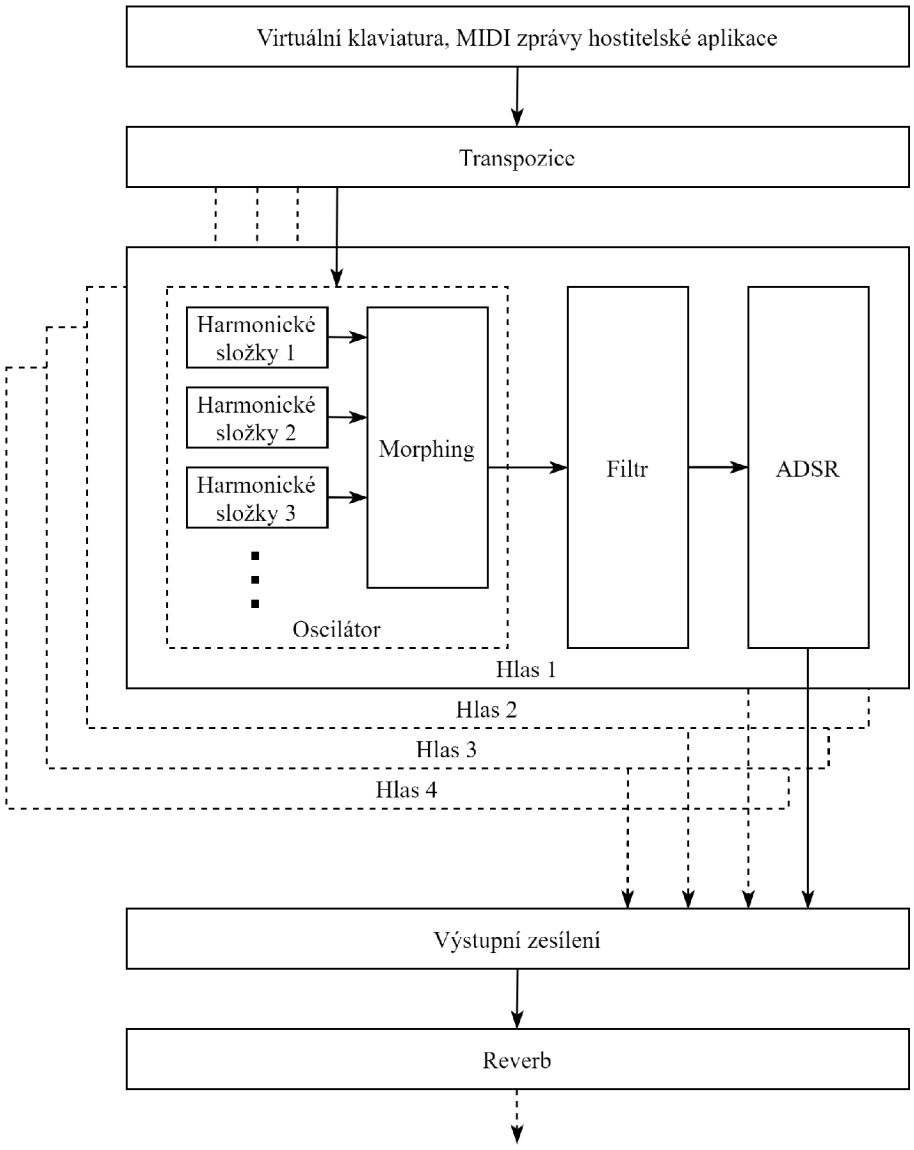
Obr. 4.1: Princip prolínání harmonických složek v čase

Grafické rozhraní

Aplikace bude využívat ovládací prvky běžné u většiny softwarových syntezátorů. Obvykle vychází z předlohy skutečných fyzických zařízení, patří mezi ně především virtuální tahové a otočné potenciometry. Náhled na uživatelem vytvořené stavy harmonických složek a přepínání mezi nimi bude využívat tzv. viewport, který umožní tažením kurzoru přesouvat náhled mezi jednotlivými obsaženými komponentami. Zobrazení hodnot většiny parametrů bude realizováno pomocí textového pole, které se zobrazí po přejetí kurzorem myši nad ovládacím prvkem. Pro možnost hry na nástroj přímo v okně aplikace bude součástí rozhraní také virtuální klaviatura. Základní umístění jednotlivých modulů v grafickém rozhraní je uvedeno na Obr.4.3.

Ostatní funkce

Většina ovládacích prvků, resp. jejich hodnoty budou implementovány jako tzv. VST parametry, které v souvislosti se zvukovými zásuvnými moduly mají význam hodnot, se kterými může pracovat i hostitelská aplikace. Parametry lze poté např.



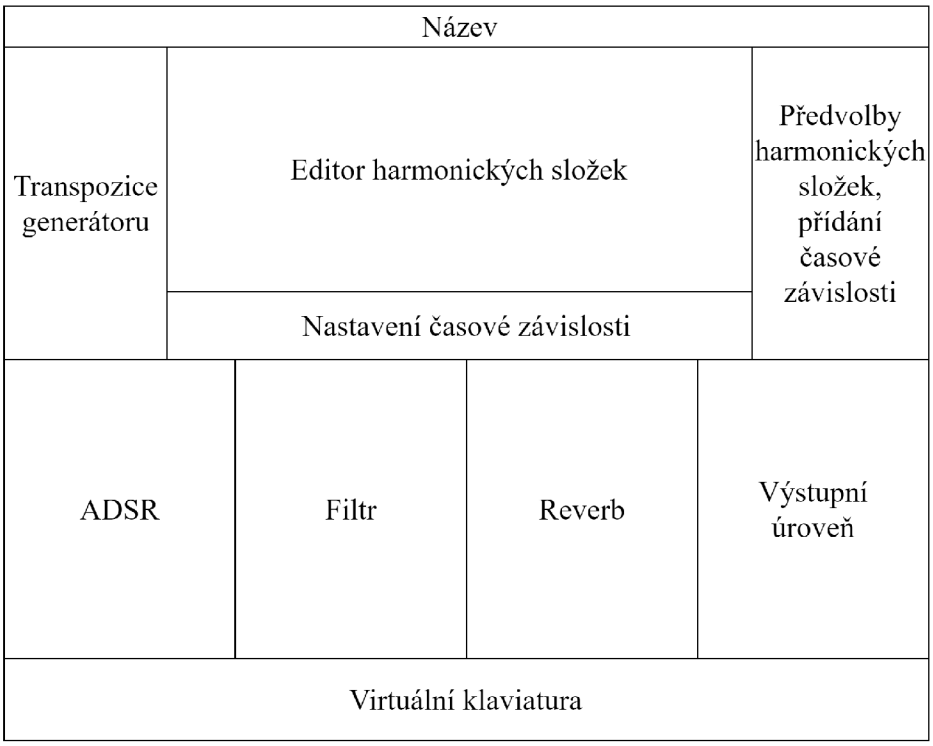
Obr. 4.2: Blokové schéma syntezátoru

automatizovat (tzn. vytvořit časovou závislost) nebo ukládat jejich aktuální hodnoty do presetu. Aplikace také tímto způsobem bude vytvořený zvuk schopna uložit a později znovu vyvolat.

Popis použití výše uvedených specifikací ve výsledné aplikaci je obsažen v příloze A.

4.2 Virtual Studio Technology

VST vytváří rozhraní mezi hostitelskou aplikací a zásuvným modulem (plug-in), v rámci kterého lze v obou směrech přenášet zvuková data a MIDI zprávy. V oblasti



Obr. 4.3: Základní návrh struktury grafického rozhraní

číslicového zpracování zvukových signálů patří tato technologie mezi nejrozšířenější formáty zásuvných modulů.

Softwarový balíček poskytující knihovny pro implementaci VST rozhraní do hostitelské aplikace nebo tvorbu zásuvných modulů je součástí tzv. VST *Software Development Kit* (SDK), který je distribuován společností Steinberg. [3]

Zásuvný modul realizovaný pomocí VST SDK je vázán na technologii VST, která musí být podporována také hostitelskou aplikací, aby bylo možné modul používat. Přestože ve zvukových aplikacích patří tento formát mezi nejvíce rozšířené a nabízí podporu pro všechny běžně používané operační systémy, existují alternativy typu *Audio Unit* (AU)[1] nebo *Advanced Audio eXtension* (AAX)[2]. Tato práce proto využívá framework *JUCE* (viz kap. 4.3), který umožňuje sestavit aplikaci ve všech uvedených formátech.

[1]Technologie vyvinutá společností Apple, exkluzivně pro vlastní operační systémy MacOS a iOS.

[2]Proprietární formát zásuvných modulů vytvořený společností Avid, který je využíván zásuvnými moduly produkčního systému Pro Tools.

4.3 JUCE framework

JUCE je aplikační framework určený obecně pro vývoj multiplatformního softwaru v jazyce C++, oproti jiným vývojářským nástrojům, které běžně obsahují funkce pro tvorbu grafického rozhraní, správu souborů, přístup k síti, kryptografii, atd., nabízí také moduly specializované na zpracování zvuku umožňující např. přístup ke zvukovým rozhraním, práci se zvukovými a MIDI daty nebo zpracování standardních formátů zvukových souborů.

JUCE framework podporuje většinu běžně používaných desktop (Windows, MacOS a Linux) nebo mobilních (Android, iOS) platforem. Z jedné verze zdrojového kódu lze sestavit aplikaci pro všechny tyto podporované platformy. Pro tvorbu zásuvných modulů jsou k dispozici tzv. *wrappery*, díky kterým vývoj aplikace různých formátů zásuvných modulů probíhá v rámci jednoho společného rozhraní. Sestavení aplikace v konkrétním formátu zásuvného modulu poté pouze napojí funkce rozhraní na funkce specifické pro daný formát. [9, 10]

4.3.1 Pomocná aplikace Projucer

Pro správu projektů založených na frameworku *JUCE* se používá pomocná aplikace *Projucer*. Jedním z hlavních účelů tohoto nástroje je zjednodušení přenosu projektů mezi různými operačními systémy. V rámci aplikace *Projucer* lze generovat zdrojové soubory pro projekty různých integrovaných vývojových prostředí (IDE), na systému Windows např. typicky nástroj *Visual Studio*, pro MacOS obvykle *Xcode*.

Dále aplikace poskytuje základní šablony pro specifický typ vytvářeného programu, je možné zvolit si např. konzolovou aplikaci, dynamickou nebo statickou knihovnu, aplikaci s grafickým rozhraním či zásuvný modul (na které je založena tato práce). [17]

4.4 Musical Instrument Digital Interface (MIDI)

MIDI je standardizované digitální rozhraní určené pro komunikaci mezi elektronickými hudebními nástroji. V rámci komunikace nedochází k přenosu zvukového signálu, ale pouze řídících dat (tzv. událostí). Tato data jsou tvořena osmibitovými bloky, které se označují jako tzv. MIDI zprávy. Ty lze rozdělit na tzv. *kanálové zprávy* a *systémové zprávy*. Kanálová data obsahují informaci, na kterém ze 16 možných virtuálních kanálů probíhá komunikace. Systémová data jsou oproti tomu určena pro všechny kanály.

MIDI zpráva je tvořena jedním tzv. *stavovým bytem* a několika *datovými bajty*. Počet datových bytů je specifický pro danou MIDI zprávu. U syntezátorů se nejčas-

těji využívají MIDI zprávy *Note On* a *Note Off*, které v datových bytech přenášejí informaci o čísle stisknuté noty a rychlosti stisku klávesy. Dále syntezátory obvykle nabízí možnost modulace tónu v průběhu generování pomocí MIDI zprávy *Pitch Bend Change* (ohýbání tónu) nebo tzv. *Modwheel* kontroleru, kterým lze řídit vybraný parametr, např. mezní frekvenci filtru. [15]

4.5 Struktura programu

Základní struktura programu je vytvořena pomocí šablony pro zásuvný modul poskytované aplikací *Projucer*, v té byl dále nakonfigurován formát sestavení VST3 a vygenerovány projekty pro vývojová prostředí *Visual Studio 2019* a *CLion*, pomocí kterých bylo možné aplikaci vyvíjet a sestavovat na platformě Windows a MacOS.

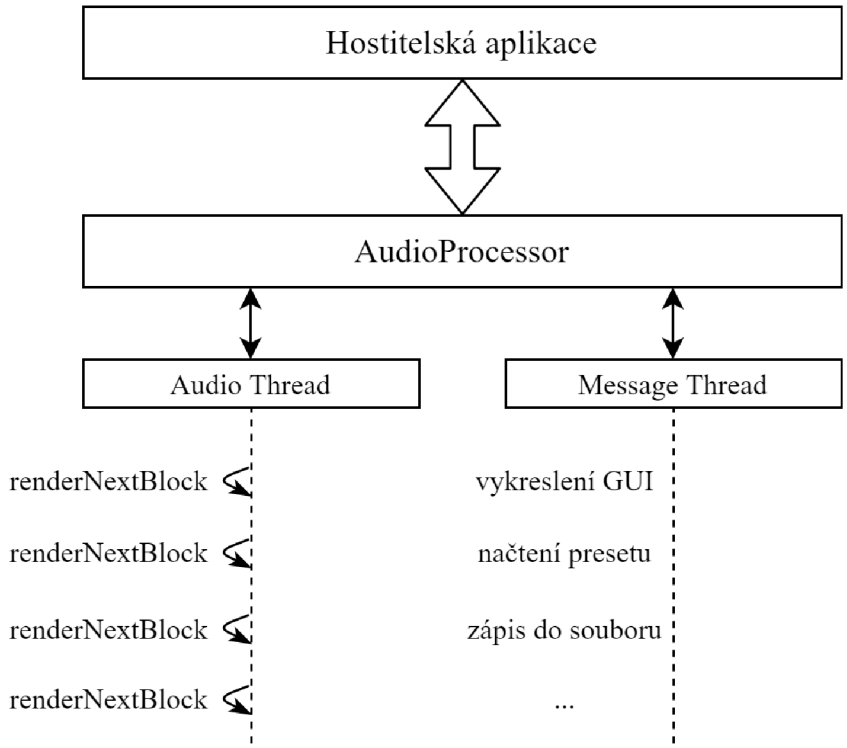
V šabloně zásuvného modulu je připraven potomek abstraktní třídy `AudioProcessor`, určené primárně pro zpracování signálu, a potomek třídy `AudioProcessorEditor`, který je určen pro grafické rozhraní. Automaticky vygenerovaný kód obsahuje základní implementace všech čistě virtuálních metod (nacházející se především ve třídě typu `AudioProcessor`) a velmi jednoduché okno pro grafické rozhraní, aplikaci je tedy možné sestavit a spustit.

Třída `AudioProcessor` kromě zpracování signálu dále zajišťuje také inicializaci komunikace s hostitelskou aplikací (např. nastavení vzorkovací frekvence při otevření nové instance zásuvného modulu), provede příslušné operace pokud hostitelská aplikace vyžaduje načtení či uložení presetu nebo uchovává parametry zásuvného modulu (viz kap. 4.1). Třída pro grafický editor (`AudioProcessorEditor`) je zpravidla určena pouze jako ovládací prostředek pro změnu parametrů. Hostitelská aplikace může editor během využívání zásuvného modulu libovolně vytvářet či mazat, hlavní funkce aplikace je tedy nutné implementovat nezávisle na tomto editoru.[10]

Sdílení dat mezi vlákny

Pro zpracování signálu je ve třídě `AudioProcessor` určena metoda `renderNextBlock`, její exekuce probíhá na samostatném vlákně (tzv. *Audio Thread*). Ostatní funkce aplikace, včetně vykreslování grafického rozhraní, jsou vykonávány na vlákně, které se v *JUCE* frameworku označuje jako tzv. *Message Thread* (viz Obr.4.4) a má zpravidla oproti vláknu pro zvuková data nižší prioritu. [10, 11]

Obě vlákna většinou sdílí datové struktury, ve kterých jsou uchovány parametry zásuvného modulu. *Message Thread* obvykle aktualizuje hodnoty parametrů pokud dojde k manipulaci s ovládacími prvky v grafickém rozhraní a zvukové vlákno aktuální hodnoty parametrů vyčítá a využívá je v rámci algoritmu pro zpracování signálu. Přístup k těmto sdíleným datovým strukturám je nutné synchronizovat,



Obr. 4.4: Základní rozdělení úloh v zásuvném modulu

aby nedošlo k souběhu – chybě, při které vlivem zpracování sdílených dat současně více vlákny mohou vznikat nesprávné hodnoty parametrů. Pro synchronizaci přístupu je v případě aplikací zpracovávajících zvuk v reálném čase nežádoucí využít prostředky, které by mohly na předem neurčitou dobu přerušit exekuci vlákna určeného pro zpracování zvuku a v důsledku toho způsobit nespojitosti (praskání) ve výstupním signálu.

Z tohoto důvodu není vhodné používat např. synchronizační primitivum *zámek*, upřednostňuje se naopak využití tzv. atomických instrukcí. V jazyce C++ verze 11 a vyšší se k tomuto účelu doporučují využít šablonové datové typy `std::atomic` ze standardní knihovny (*STL*), které zaručují synchronizaci přístupu ke sdíleným datům a pro jednoduché datové typy nebo ukazatele na komplexnější datové typy (u většině běžně používaných platforem) nevyužívají k synchronizaci zámky. Nevzniká tak riziko, že bude blokováno vlákno pro zpracování zvuku. [11, 13, 14]

V metodách, které mají být provedeny na vlákně určeném pro zpracování zvuku by se dále měly vykonávat pouze operace, u kterých je možné předem odhadnout dobu vykonávání. Pokud nelze zaručit, že čas operace při každém běhu aplikace nepřesáhne požadovanou hodnotu, vzniká teoreticky možnost nespojitostí ve výstupním signálu. Z tohoto důvodu např. není vhodné dynamicky alokovat paměť, provádět zápis nebo čtení souborů, volat funkce operačního systému nebo obecně

používat třídy či metody, které by mohly mít tyto funkce ve své implementaci. [12]

Datové struktury pro bezpečné sdílení mezi více vlákny nebo manipulaci s daty bez využití zámku jsou k dispozici také v *JUCE* frameworku. Splnění případně nesplnění těchto kritérií je obvykle explicitně uvedeno v dokumentaci. Při realizaci aplikace v rámci této práce byly některé z poskytovaných datových struktur použity pro uchování parametrů zásuvného modulu, přístup k hodnotám parametrů z vlákna pro zpracování zvuku byl zajištěn pomocí standardních C++ atomických datových typů.

4.6 Programové řešení syntezátoru

Základní požadavky na syntezátor obecně zahrnují mimo generováním signálu také funkce řídící parametry generátorů na základě přijatých MIDI zpráv. Pokud je daný syntezátor polyfonní, musí dále existovat mechanismus, který zajistí správu jednotlivých hlasů, zavolá po přijetí odpovídající MIDI zprávy metodu generující požadovaný průběh a ve správný okamžik generování přeruší, případně vykoná určitou akci, když není k dispozici žádný volný hlas.

Uvedené funkce jsou obvykle nezbytnou součástí většiny syntezátorů, lze je tedy implementovat do obecnější struktury aplikovatelné na libovolnou realizaci syntezátoru. V JUCE frameworku byly k tomuto účelu vytvořeny třídy `Synthesiser`, `SynthesiserVoice` (dále v textu též označovaný jako hlas) a `SynthesiserSound`.

4.6.1 Třída Synthesiser

Třída `Synthesiser` reprezentuje abstrakci hudebního zařízení, které na základě přijatých MIDI zpráv řídí objekty zajišťující generování signálu. Mezi klíčové metody, které v rámci třídy řeší problematiku zmíněnou v úvodu této podkapitoly patří

`renderNextBlock` předává zvuková data mezi aktuální instancí třídy `Synthesiser` a ostatními objekty v aplikaci, zajišťuje naplnění bloků signálem odpovídajících hlasů, volání jednotlivých hlasů probíhá v oddělených privátních metodách

`noteOn` volána po přijetí MIDI zprávy *Note On*, v případě potřeby může pro generování tónu poslední stisknuté klávesy zprostředkovat volný hlas

`noteOff` volána po přijetí MIDI zprávy *Note Off*, přeruší generování signálu pro danou notu a uvolní používaný hlas

Generování samotného průběhu signálu uvnitř třídy neprobíhá, signál vzniká v instancích třídy `SynthesiserVoice`, které jsou privátními datovými členy objektu typu `Synthesiser`, do kterého musí být přidány pomocí metody `addVoice` před tím, než se začnou generovat zvuková data. [10]

Třída SynthesiserSound

Třída slouží jako pomocná struktura umožňující omezit hlasy třídy `Synthesiser` na konkrétní MIDI kanály nebo MIDI noty, v rámci rozsahu klaviatury nebo specifických MIDI kanálů je tak možné generovat odlišný výstupní signál. Celá tato funkcionalita je zajištěna pomocí dvou členských metod

`appliesToNote` rozhoduje jestli vstupní číslo MIDI noty může spustit generování signálu

`appliesToChannel` rozhoduje jestli data přicházející z daného MIDI kanálu spustí generování signálu

Ve výsledné aplikaci se využívaly stejné typy hlasů pro všechny MIDI noty i kanály. Minimálně jedna instance třídy typu `SynthesiserSound` je však nutnou součástí objektu `Synthesiser`. K tomuto účelu byl vytvořen potomek této třídy, který v uvedených metodách generování signálu žádným způsobem neomezoval. [10]

Třída SynthesiserVoice

V této třídě jsou implementovány metody pro generování průběhu konkrétní instance typu `SynthesiserSound`. Reprezentuje hlas syntezátoru, který objekt `SynthesiserSound` může použít. Dále se zde nachází funkce určené k řízení průběhu generování, které, zpravidla na základě vstupních MIDI zpráv, upravují parametry generátoru (např. změnu amplitudy podle obálky ADSR).

Většina obsažených metod je čistě virtuálních, k vytvoření instance tohoto typu je proto nutné použít potomka třídy `SynthesiserVoice` a v něm tyto metody definovat. [10]

Klíčové metody zajišťující generování a řízení jeho průběhu jsou

`renderNextBlock` provádí naplnění vstupního bloku zvukovými daty daného hlasu

`startNote` volána po příjetí MIDI zprávy *Note On*, využívána zpravidla pro nastavení frekvence generátoru podle přijaté noty, případně uvedení amplitudové obálky do počátečního fáze

`stopNote` volaná po přijetí MIDI zprávy *Note Off*, poskytuje možnost definovat způsob ukončení generování (např. spustit u obálky ADSR fázi doznění) a ve vhodný okamžik (např. při poklesu doznívajícího signálu pod určitou úroveň) hlas uvolnit k dalšímu použití.

Pro generování správné frekvence odpovídající standardnímu temperovanému ladění je důležité volat při změně vzorkovací frekvence hostitelské aplikace metodu `setCurrentPlaybackSampleRate`, kterou se nová hodnota předá hlasu a jsou podle ní upraveny parametry generátoru.

4.7 Oscilátor

Jádro úlohy generování signálu tvoří periodické volání funkce, která na základě aktuální fáze poskytuje okamžitou hodnotu určitého průběhu. Výpočet této hodnoty lze realizovat pomocí matematického vztahu nebo s využitím tabulky (tzv. *lookup table*).

V rámci této práce byly vytvářeny generátory harmonického signálu, uvedené příklady se tedy budou zabývat tímto průběhem, obecně jsou však metody aplikovatelné také na jiné typy signálu.

Klíčovými parametry pro vytvoření průběhu je frekvence, aktuální fáze, přírůstek fáze a vzorkovací kmitočet. Přírůstek fáze pro harmonickou funkci *sin* je určen vztahem

$$incr_{sin} = f \cdot \frac{2\pi}{f_{vz}} \tag{4.1}$$

kde f určuje frekvenci generovaného signálu a f_{vz} vzorkovací frekvenci systému, v rámci kterého generování probíhá (např. tedy hostitelské aplikace). Pokud by systém pracoval s rozdílnou vzorkovací frekvencí, nebylo by z hlediska hudebního nástroje dodrženo standardní ladění s referencí *440 Hz*.

Aktuální hodnota fáze se uchovává v samostatné proměnné a po každé iteraci je zvětšena o fázový přírůstek. Z hlediska softwarové implementace je vhodné omezit příliš velké argumenty funkce `sin`, které můžou u některých C++ kompilátorů způsobit výraznou odchylku. Po dosažení hodnoty fáze větší než jedna perioda funkce *sin* je proto vhodné aktuální fázi snížit o hodnotu 2π. [16]

V následujícím výpisu je uvedena celková implementace využívající harmonickou funkci *sin* pro naplnění bloku zvukových dat.

Výpis 4.1: Příklad implementace harmonického generátoru pomocí funkce `sin`.

```
#define TWO_PI (6.283185307)

/* Vstupní a pomocné proměnné. */
float frequency = 440.0f, currentPhase = 0.0f, phaseIncrement;
float currentSampleRate = 44100;
float amplitude = 0.5f

/* Metoda naplní celý vstupní blok
   hodnotami harmonického signálu. */
void renderNextBlock(float* audioBuffer, int numSamples)
{
  phaseIncrement = TWO_PI * frequency / currentSampleRate;

  for (int sample = 0; sample < numSamples; ++sample)
  {
    float outputValue = std::sin(currentPhase) * amplitude;

    currentPhase += phaseIncrement;
    if (currentPhase >= TWO_PI) currentPhase -= TWO_PI;

    audioBuffer[sample] = outputValue;
  }
}
```

Hlavní nedostatek výše uvedeného kódu z hlediska zpracování v reálném čase představuje počítání funkce `sin`, který může vyžadovat nezanedbatelné množství výpočetního času, prakticky je tedy vhodnější využít pro generování tabulku.

Tabulkový generátor redukuje čas výpočtu použitím již připravené sady funkčních hodnot jedné periody daného průběhu. Výsledný signál vzniká převzorkováním tabulky zpravidla nižší vzorkovací frekvencí, perioda nové vzorkovací frekvence závisí na frekvenci generovaného signálu a je určena vztahem

$$incr_{tab} = f \cdot \frac{N_{tab}}{f_{vz}} \tag{4.2}$$

kde f je frekvence výstupního signálu, N_{tab} počet funkčních hodnot v tabulce a f_{vz} vzorkovací frekvence systému.

Pokud se během převzorkování neprochází tabulkou s přírůstkem celého počtu vzorků, je nutné hodnotu „mezi vzorky“ určit interpolací nebo ji zaokrouhlit na

nejbližší vzorek. Pro lepší zvukovou kvalitu se obvykle využívá lineární nebo kvadratická interpolace, zaokrouhlením může dojít k vyšší odchylce, je však méně náročné na systémové prostředky. [16]

Samotnému výpočtu funkčních hodnot předchází naplnění tabulky jednou periodou harmonického signálu, což představuje pouze mírnou modifikaci již zmíněného generování pomocí funkce `sin`, vzorová implementace vytvoření tabulky je uvedena v následujícím výpisu.

Výpis 4.2: Vytvoření tabulky pro harmonický generátor.

```
#define TWO_PI (6.283185307)

int tableSize = 1024;
float lookupTable[tableSize];
float currentPhase = 0.0f, phaseIncrement;

/* Naplnění tabulky */
phaseIncrement = TWO_PI / tableSize;
for (int i = 0; i < tableSize; ++i)
{
  lookupTable[i] = std::sin(currentPhase);
  currentPhase += phaseIncrement;
}
```

Obecně je možné tabulku naplnit periodou libovolného průběhu, pokud však obsahuje také vyšší harmonické složky, může při generování (především vysokých frekvencí) vznikat aliasing. [8]

Implementace výpočtu z tabulky s je uvedena v následujícím výpisu, metoda využívá zaokrouhlování k nejbližšímu nižšímu vzorku. Oproti implementaci s využitím funkce `sin` je v tomto případě klíčové hlídat hodnotu pozice v tabulce, aby nedošlo k přístupu mimo hranice pole, čímž by mohlo dojít k pádu programu. [16]

Výpis 4.3: Příklad implementace harmonického generátoru pomocí tabulky.

```
#define TWO_PI (6.283185307)

/* Vstupní a pomocné proměnné. */
int tableSize = 1024;
float lookupTable[tableSize];
float tablePosition = 0.0f, tableIncrement;
float frequency = 440.0f;
float currentSampleRate = 44100;
float amplitude = 0.5f;
```

```
/* Metoda naplní celý vstupní blok
   hodnotami z tabulky. */
void renderNextBlock(float* audioBuffer, int numSamples)
{
  tableIncrement = frequency * tableSize / currentSampleRate;

  for (int sample = 0; sample < numSamples; ++sample)
  {
    float outputValue = lookupTable[(int) tablePosition];
    audioBuffer[sample] = outputValue * amplitude;

    tablePosition += tableIncrement;
    if (tablePosition >= tableSize) tablePosition -= tableSize;
  }
}
```

4.8 Generátory obálek

Generátor obálky je založen na interpolaci mezi známými hodnotami, které jsou odděleny definovaným časovým rozestupem. Vstupní data generátoru tvoří sada po sobě jdoucích (často uměle vytvořených) okamžitých hodnot vybraného parametru a údaje o době přechodu mezi jednotlivými hodnotami.

Na základě těchto dat se vytvoří průběh, který počtem vzorků při dané vzorkovací frekvenci odpovídá době trvání obálky, a pomocí kterého je následně možné v reálném čase modulovat požadovaný parametr. Vytvořený průběh lze uchovávat v tabulce, ze které jsou hodnoty vyčítány podobně jako u signálových generátorů v předchozí sekci nebo se hodnoty generují výpočtem matematického vztahu.

Standardně se pro generátor obálky využívá lineární interpolace, která mezi okamžitými hodnotami vytváří průběh popsaný lineární funkcí. Pro případ, který uvažuje interpolaci se dvěma hodnotami platí pro libovolnou hodnotu $y(t)$ v době trvání obálky vztah

$$y(t) = y(t_1) + t \cdot \frac{y(t_2) - y(t_1)}{t_2 - t_1} \tag{4.3}$$

kde $y(t_1)$ a $y(t_2)$ odpovídají počáteční a koncové hodnotě obálky a t_1 a t_2 společně určují dobu trvání obálky. Časové parametry v sekundách je nutné při implementaci v číslicovém systému převést na počet vzorků d_x vztahem

$$d_x = t_x \cdot f_{vz} \tag{4.4}$$

kde f_{vz} je aktuální vzorkovací frekvence a t_x časový údaj v sekundách.

Implementace obecného generátoru obálky využívajícího lineární interpolaci je uvedena v následujícím výpisu, vně výpočetní metody jsou deklarovány základní parametry obálky, proveden převod času v sekundách na počet vzorků a vytvořena proměnná pro ukládání aktuální pozice generátoru.

Výpis 4.4: Příklad implementace obálky pomocí lineární interpolace. [16]

```
float sampleRate = 44100;
float durationSeconds = 0.5;
int durationSamples = (int) (durationSeconds * sampleRate);
int currentPosition = 0;

/* Metoda vrací vzorek obálky v daném čase. */
float getNextLinearEnvelopeSample  (float startValue,
                                    float durationSamples,
                                    float stopValue,
                                    int* currentPosition)
{
  if ((*currentPosition)++ < durationSamples)
  {
    float frac = *currentPosition / durationSamples;
    return startValue + (stopValue - startValue) * frac;
  }
  else
    return stopValue;
}
```

Programové řešení obálky typu ADSR (viz. kap. 1.1) pouze rozšiřuje předchozí ukázku interpolace o další hodnoty a podmíněné příkazy, kterými se kontroluje aktuální fáze obálky a spouští odpovídající generátor průběhu. [16]

5 Implementace aplikace

Tato kapitola je zaměřena na rozbor konkrétních softwarových řešení aplikace a implementaci konceptů z předchozí kapitoly do výsledného programu (v textu též označovaný jako *Additive*). V průběhu psaní programu byl využíván převážně objektově orientovaný přistup, následující sekce tedy klade důraz na souhrnný rozbor hlavních tříd a jejich vzájemných vztahů. Bude shrnuta vrstva zajišťující zpracování signálu a vrstva grafického rozhraní, dvě hlavní a do značné míry nezávislé součásti aplikace. Detailní seznam všech tříd vytvořených v rámci této práce je spolu se stručným popisem jejich funkce uveden v příloze B.

5.1 Sekce zpracování signálu

Všechny třídy pro zpracování signálu jsou zapouzdřeny ve třídě `PluginProcessor`, která je vytvořena automaticky v šabloně zásuvného modulu a je potomkem bázové třídy `AudioProcessor` (viz kap. 4.5). Na úrovni této třídy se nachází hlavní metoda pro předávání bloků zvukových a MIDI dat mezi hostitelskou aplikací a zásuvným modulem – `processBlock`, dále jsou zde pomocné funkce pro identifikaci zásuvného modulu, nastavení zvukových kanálů, navázání na grafické rozhraní nebo ukládání a načítání presetů.

Důležitým privátním členem je instance třídy `AudioProcessorValueTreeState`, která uchovává parametry zásuvného modulu, současně umožňuje uložení a načtení stavu všech parametrů ve formátu *XML*. Parametry vytvořené v tomto objektu jsou zároveň zpřístupněny hostitelské aplikaci pro automatizaci. Referenci na danou instanci také využívají všechny objekty, které potřebují přistupovat k hodnotám parametrů. V grafickém editoru jsou to ovládací prvky, které změnou polohy zapisují do parametrů novou hodnotu. Třídy pro zpracování signálu čtou aktuální hodnoty parametrů a případně podle nich upravují proces generování zvukových dat.

Třída AdditiveAudioEngine

Pro lepší oddělení funkčních bloků kódu a díky tomu i jednodušší implementaci případných rozšíření programu byly všechny objekty zajišťující generování signálu a zpracování efekty zapouzdřeny do třídy `AdditiveAudioEngine`.

Předávání zvukových a MIDI dat probíhá skrze metodu `renderNextBlock`, která je volána z třídy `PluginProcessor` a tato data jsou dále předána generátoru a efektům.

Generování signálů podle vstupních MIDI zpráv řídí privátní datový člen typu `AdditiveSynthesiser`, který je potomkem třídy `Synthesiser` (viz kapitola 4.6.1).

K následné úpravě signálu efektem dozvuku využita instance typu `juce::dsp::Reverb`.

Třída AdditiveSynthesiserVoice

Třída `AdditiveSynthesiserVoice` je potomkem abstraktní třídy `SynthesiserVoice`, obsahuje metody pro řízení generátoru signálu v rámci jednoho hlasu syntezátoru (viz kap. 4.6.1).

Po přijetí MIDI zprávy *Note On* je v této třídě volána metoda `startNote`, která nastaví frekvenci generátoru, spustí fázi *attack* obálky ADSR a následně jsou metodě `renderNextBlock` předána zvuková data, ke kterým je přidán vygenerovaný signál. Zpráva *Note Off* vyvolá metodu `stopNote`, v té se ADSR přepne na fázi *release*. Samotná obálka je implementována v privátním členu `juce::ADSR`.

5.1.1 Třída AdditiveOscillator

Tato třída odděluje proces generování od třídy `AdditiveSynthesiserVoice`. Generátor využívá tabulky, které jsou vytvořeny v konstruktoru třídy. Z důvodu optimalizace byla vytvořena tabulka pro každou harmonickou složku, během generování je tak možné používat jeden společný údaj o fázi. Při výpočtu vzorku signálu není nutné měnit fázi jednotlivě pro každou složku, čímž se ušetří poměrně významný počet operací součtu a podmíněných příkazů kontrolujících, zda je hodnota fáze v rozsahu $0 - 2\pi/f_{vz}$ (viz kap. 4.7).

Metody pro generování harmonických složek

Proces generování začíná v metodě `renderNextBlock`, které je předána reference na blok zvukových dat zapouzdřených v šablonové třídě `AudioBuffer<float>`. Podle přijatých MIDI zpráv *Note On* a *Note Off* může být generování omezeno pouze na určitý počet vzorků v bloku (v průběhu daného bloku mohlo např. dojít k uvolnění klávesy). K vymezení tohoto úseku jsou využity vstupní parametry celočíselného typu `startSample` a `numSamples`. V tomto omezeném úseku se následně pro každý vzorek volá metoda `getNextTableSample()`, která vrací aktuální hodnotu signálu vytvořeného součtem jednotlivých harmonických složek. Poté je vypočtena aktuální hodnota generátoru obálky, kterou je upravena úroveň výstupního vzorku. Takto upravený vzorek je přičten k obsahu odpovídajícího vzorku ve vstupním bloku zvukových dat. Stejný vstupní blok dat může být následně využíván také ostatními hlasy syntezátoru (pokud je stisknuto více kláves současně), je proto nutné vzorky již obsažené v daném bloku ponechat v původním stavu a pouze k nim přidat (přičíst) nově vygenerovaný signál.

Časová závislost harmonických složek

Jednotlivé sady harmonických složek, mezi kterými může signál prolínat, jsou zapouzdřeny ve třídě `HarmonicValuesSet`. Kromě úrovní harmonických je zde také člen pro uchování času interpolace mezi aktuálními a následujícími složkami.

Samotná implementace prolínání harmonických se nachází v metodě pro generování `getNextTableSample()`. Aktuální hodnota úrovně harmonické je určena lineární interpolací mezi jednotlivými složkami. Dobu interpolace udává parametr nastavitelný v grafickém rozhraní. Výchozí jednotkou parametru času prolínání jsou milisekundy, které se pro samotný výpočet převádí na počet vzorků odpovídající aktuální vzorkovací frekvenci. V metodě je dále obsažen kód pro přechod na další sadu harmonických složek v případě, že je byla dosažena doba prolínání a v aplikaci jsou vytvořené další stavy, mezi kterými je možné úrovně harmonických interpolovat.

Filtrace signálu

Každý hlas má k dispozici vlastní modul filtru, čímž bylo možné implementovat sledovač klaviatury. Objekt pro filtraci je zapouzdřen ve třídě `AdditiveOscillator` a je tvořen typem `juce::dsp::StateVariableFilter::Filter`, který je realizován na principu filtru se stavovou kanonickou strukturou. Výpočet koeficientů filtru se provádí pouze pokud dojde skrze grafické rozhraní nebo hostitelskou aplikaci ke změně některých parametrů filtru. Většinu času generování signálu jsou parametry filtrů nezměněny, tímto způsobem je tedy možné ušetřit značné množství nadbytečných výpočtů.

Optimalizace s využitím instrukcí typu SIMD

S cílem zrychlení exekuce funkce `getNextTableSample()`, která podle výsledků profilování aplikace spotřebovala většinu z celkového času stráveného v hlavní metodě pro zpracování zvukových dat, byla experimentálně implementována druhá verze této metody s využitím instrukcí typu SIMD. Zpracování dat pomocí SIMD umožňuje v rámci jednoho instrukčního cyklu provést aritmetickou operaci nad více operandy současně. V běžně rozšířených instrukčních sadách s podporou SIMD lze obvykle pracovat alespoň se 4 operandy datového typu čísla s plovoucí řádovou čárkou. V *JUCE* frameworku je pro tento typ paralelního zpracování k dispozici šablonová třída `juce::dsp::SIMD`, ve které jsou zapouzdřeny konkrétní implementace SIMD instrukcí pro všechny frameworkem podporované platformy.

Samotná optimalizace se použila pro úsek kódu, ve kterém se vyčítá signál harmonických z tabulky a počítá prolínání více stavů složek pomocí lineární interpolace. V původním sekvenčním přístupu tento výpočet probíhá v rámci standardního `for`

cyklu jednotlivě pro každou složku, s instrukcemi SIMD byl záměr tyto operace provádět paralelně pro 4 harmonické složky.

Před paralelním výpočtem bylo nutné data přesunout do registrů určených pro tento typ instrukcí, tím se však více projevila režie způsobená častější manipulací s daty v paměti. Přestože aritmetické operace následně mohly proběhnout rychleji, výsledná doba výpočtu celé funkce pro generování se nakonec oproti sekvenčnímu zpracování v podstatě nezměnila. Finální implementace tedy signál generuje pomocí původní sekvenčně vykonávané metody. Optimalizovaná verze je ve zdrojovém kódu uvedena pouze jako komentář, protože v průběhu vývoje přestala být kompatibilní s novými úpravami.

Další možností optimalizace by mohla být komplexnější úprava algoritmu pro generování signálu tak, aby se maximalizoval počet operací nad daty, které jsou již jednou uloženy do registrů. [18]

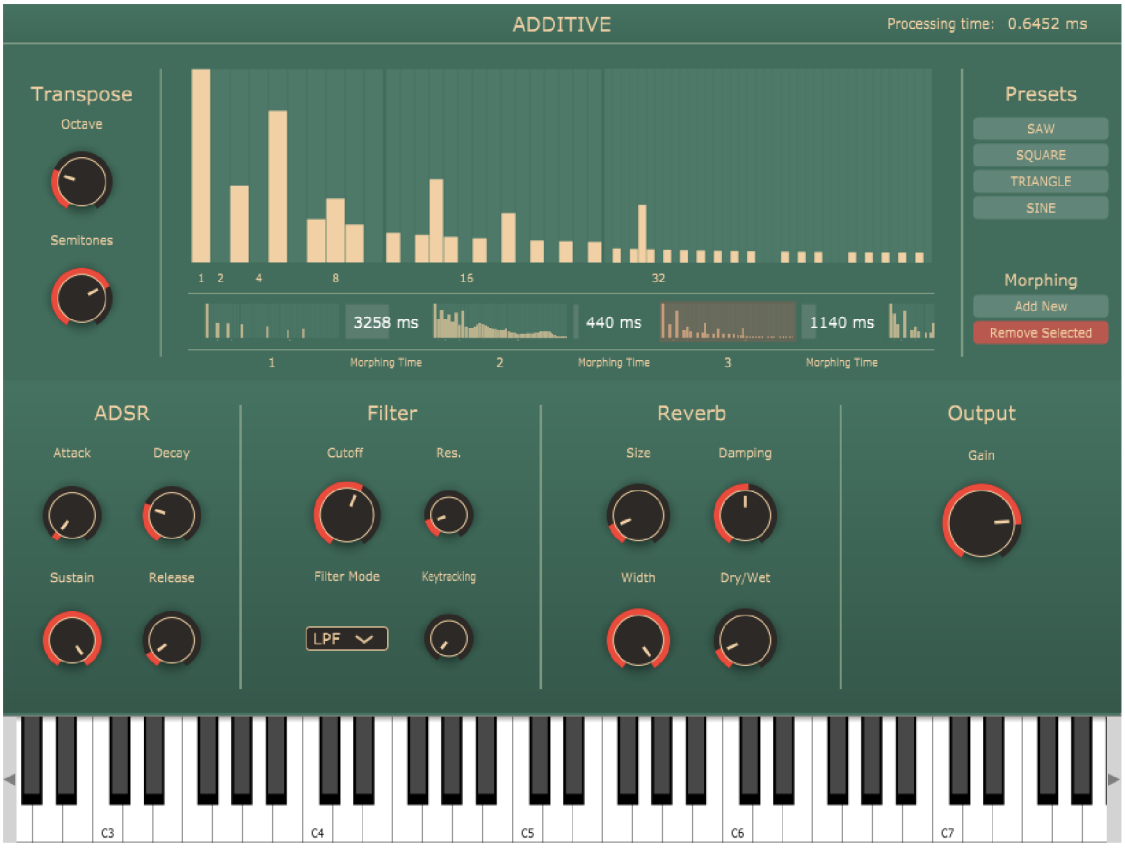
5.2 Sekce grafického uživatelského rozhraní

Pro implementaci grafického rozhraní je v šabloně zásuvného modulu vyhrazena třída `PluginEditor`, její instance se během otevření zásuvného modulu v hostitelské aplikaci naváže na objekt typu `PluginProcessor`. `PluginEditor` je potomkem třídy `Component`, která ve frameworku *JUCE* obecně reprezentuje prostor určený k vykreslení grafického rozhraní, do kterého je možné vkládat další komponenty, kterými mohou být např. ovládací prvky nebo vektorové či bitmapové obrazce. Každá třída odvozená od typu `Component` implementuje metody, které umožňují nastavit velikosti a umístění obsažených komponent. [10]

Struktura grafického rozhraní aplikace Additive

Výsledné grafické rozhraní je na Obr.5.1, hlavní součásti tvoří horizontálně orientované sekce pro nastavení generátoru harmonických složek, ovládací prvky pro úpravu signálu a virtuální klaviatura. V horní části je dále úzký pruh s názvem aplikace, který obsahuje také časovou indikaci udávající dobu zpracování signálu od získání zvukového bloku z hostitelské aplikace po navrácení bloku zpět do hostitelské aplikace.

Všechny uvedené oblasti jsou vymezeny vlastní třídou typu `Component`, které dále obsahují menší komponenty zobrazující ovládací prvky.



Obr. 5.1: Náhled na aplikaci.

Závěr

Tato práce se věnovala principům tvorby hudebního signálu pomocí součtové syntézy a implementaci VST zásuvného modulu založeného na tomto přístupu.

Byly popsány a porovnány základní metody tvorby umělého zvuku a shrnuty možné přístupy k realizaci součtové syntézy.

V praktické části byl realizován zásuvný modul součtového syntezátoru založený na technologiích *VST3* a *JUCE*. Výsledná aplikace poskytuje signálový generátor s nastavitelnými poměrem 64 harmonických složek, řízením fundamentu na základě vstupních MIDI zpráv hostitelské aplikace nebo virtuální klaviatury a čtyřhlasou polyfonii. Nástroj je doplněn funkcemi pro rychlé vyvolání speciálních případů modulového kmitočtového spektra (např. obdélníkového nebo pilového signál). Modelování průběhu amplitudy signálu zajišťuje obálka typu ADSR. Časovou závislost spektrálních složek lze vytvořit lineární interpolací mezi uživatelem vytvořenými stavy harmonických.

Pro možnost dalších úprav zvukové barvy byly do aplikace přidány kmitočtové filtry typu dolní, horní a pásmová propust s nastavitelným mezním kmitočtem, činitelem jakosti a sledovačem klaviatury. K signálu je dále možné přidat také vestavěný efekt reverb.

Výstupní sekci tvoří ovladač zesílení pro omezení zkreslení, které může vznikat při hře více hlasů současně.

Zásuvný modul lze využít jako softwarový hudební nástroj ve standardních systémech *DAW* s podporou hostování formátu *VST3*. V rámci hostitelské aplikace program umožňuje načtení a uložení celkového stavu (presetu) a kontrolu parametrů skrze automatizaci.

Literatura

[1] SYROVÝ, V. *Hudební akustika*. 3., dopl. vyd. V Praze: Akademie múzických umění, 2013. Akustická knihovna Zvukového studia Hudební fakulty AMU. ISBN 978-807-3312-978.

[2] SCHIMMEL, Jiří. *Elektroakustika*. Brno: Vysoké učení technické v Brně, 2014. ISBN 978-80-214-4716-5.

[3] RUSS, M. *Sound Synthesis and Sampling*. Third Edition. Oxford: Focal Press, 2009. ISBN 978-0-240-52105-3.

[4] URBAN, O. *Instrumentář elektroakustického zvuku*. V Praze: Akademie múzických umění, 2007. Akustická knihovna Zvukového studia Hudební fakulty AMU. ISBN 978-80-7331-115-5.

[5] SMITH III, J. O. *Spectral Audio Signal Processing* [online]. W3K Publishing, 2011 [cit. 2019-11-24]. ISBN 9780974560731. Dostupné z URL: <https://www.dsprelated.com/freebooks/sasp/>.

[6] RODET, X. a P. DEPALLE. Spectral Envelopes and Inverse FFT Synthesis. *Audio Engineering Society Convention 93* [online]. 1992, **93** [cit. 2019-11-24]. Dostupné z URL: <http://www.aes.org/e-lib/browse.cfm?elib=6740>.

[7] KLINGBEIL, M. K. *Spectral Analysis, Editing, and Resynthesis: Methods and Applications* [online]. New York, 2009 [cit. 2019-12-03]. Dostupné z URL: <http://www.klingbeil.com/data/Klingbeil_Dissertation_web.pdf>. Dizertační práce. Columbia University.

[8] GEIGER, G. Table Lookup Oscillators Using Generic Integrated Wavetables. *Proc. of the 9th Int. Conference on Digital Audio Effects (DAFx-06)* [online]. 2006, 9 [cit. 2019-11-24]. Dostupné z URL: <https://bit.ly/2D7Eu6q>.

[9] ROBINSON, Martin. *Getting Started With JUCE* [online]. Birmingham: Packt Publishing, 2013 [cit. 2019-12-07]. ISBN 978-1-78328-331-6. Dostupné z URL: <https://www.packtpub.com/application-development/getting-started-juce>.

[10] JUCE Documentation. *JUCE* [online]. [cit. 2019-11-25]. Dostupné z URL: <https://docs.juce.com/master/index.html>.

[11] DOUMLER, Timur. CppCon 2015: Timur Doumler "C++ in the Audio Industry". In: *YouTube* [online]. 2015 [cit. 2020-05-28]. Dostupné z: <https://www.youtube.com/watch?v=boPEO2auJj4>

[12] GOODLIFFE, Pete. The Golden Rules of Audio Programming, Pete Goodliffe. In: *YouTube* [online]. 2016 [cit. 2019-11-17]. Dostupné z URL: <https://www.youtube.com/watch?v=SJXGSJ6Zoro&t=583s>.

[13] Souběh. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2020 [cit. 2020-05-28]. Dostupné z: <https://cs.wikipedia.org/wiki/Soub%C4%9Bh>

[14] Std::atomic. *Cppreference* [online]. [cit. 2020-05-28]. Dostupné z: <https://en.cppreference.com/w/cpp/atomic/atomic>

[15] SCHIMMEL, Jiří. *Studiová a hudební elektornika*. Brno: Vysoké učení technické v Brně, 2012. ISBN 978-80-214-4452-2.

[16] BOULANGER, R., LAZZARINI. V., *The Audio Programming Book.* Cambridge, Mass.: MIT Press, c2011. ISBN 978-0262014465.

[17] Projucer Manual. *JUCE* [online]. [cit. 2019-11-28]. Dostupné z URL: <https://juce.com/discover/stories/projucer-manual>.

[18] BIKKER, Jacco. Practical SIMD Programming. In: *Cs.uu.nl* [online]. Utrecht, 2017 [cit. 2020-05-28]. Dostupné z: <https://bit.ly/3de7g5L>

Seznam symbolů, veličin a zkratek

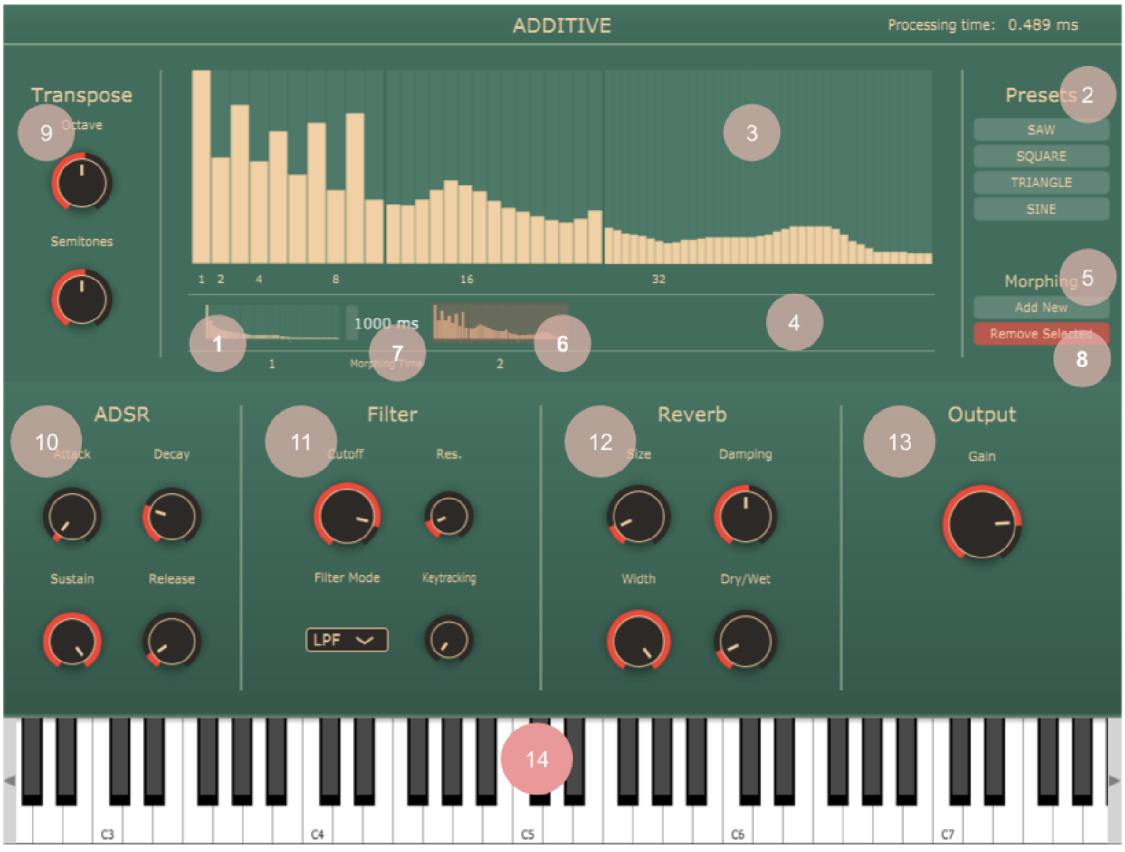
FM	frekvenční modulace
FFT	rychlá Fourierova transformace
IFFT	inverzní rychlá Fourierova transformace
STS	Short Term Spectrum
DSP	číslicové zpracování signálů – Digital Signal Processing
VST	Virtual Studio Technology
GUI	grafické uživatelské rozhraní – Graphical User Interface
DAW	Digital Audio Workstation
SDK	Software Development Kit
MIDI	Musical Instrument Digital Interface
AU	Audio Unit
AAX	Advanced Audio eXtension
IDE	Integrated Development Environment
STL	Standard Template Library
XML	Extensible Markup Language

Seznam příloh

A Popis funkcí aplikace

A.1 Oscilátor

Po načtení nové instance zásuvného modulu v hostitelské aplikaci je ve výchozím stavu připravena sada harmonických složek pro generování pilového signálu (1). Základní průběhy pro aktuálně zobrazené harmonické složky je možné vyvolat stisknutím některého z tlačítek pod nápisem *Presets* (2). Vlastní poměry lze vytvořit tažením myší v prostoru s virtuálními tahovými potenciometry (3). Pod tímto prostorem se nachází zmenšené náhledy na jednotlivé sady složek (4). Přidání nové sady se provede v sekci *Morphing* stisknutím tlačítka *Add New* (5). Hlavní náhled pro editaci složek se po stisknutí tlačítka přepne na novou sadu, indikace aktuálně vybrané sady je zajištěna pomocí světle červeného obdélníku na pozadí zmenšeného náhledu (6). Nově přidané harmonické budou mít nastaveny poměry podle poslední sady.



Obr. A.1: Popis prvků grafického rozhraní

V prostoru se zmenšenými náhledy (4) se po přidání nové sady objeví ovládací

prvek pro nastavení času prolínání mezi více stavy harmonických složek (7). Změnu tohoto parametru je možné provést vertikálním tahem myší. Při větším počtu zmenšených náhledů se lze v celé oblasti přesouvat tažením myši do stran. Aktuálně vybraná sada se odstraní stisknutím tlačítka *Remove Selected* (8).

Celková transpozice generovaného signálu o určitý počet oktáv nebo půltónu se upraví ovládacími prvky *Octave* a *Semitones* (9).

A.2 Modifikátory

Ve spodní části aplikace se nachází pruh s modifikátory signálu. V modulu *ADSR* (10) lze nastavit průběh jednotlivých fází obálky. Modul *Filter* (11) umožňuje úpravu základních parametrů filtrace (mezní frekvence – *Cutoff* a činitel jakosti – *Res.*), volbu typu filtru v rozevíracím seznamu (*LPF* – dolní propust, *BPF* – pásmová propust, *HPF* – horní propust) a nastavení sledovače klaviatury, kde maximální poloha (hodnota *1.0*) bude udržovat fixní poměr frekvence generovaného signálu na daném hlasu k mezní frekvenci filtru a minimální poloha (hodnota *0.0*) mezní frekvenci filtru ponechá na ustálené hodnotě určené ovládacím prvkem *Cutoff*.

Modulem *Reverb* (12) je možné k signálu přidat simulaci reálného poslechového prostoru, pro který lze určit velikost (*Size*), tlumení vyšších frekvencí signálu (*Damping*) nebo šířku stereofonního vjemu signálu (*Width*). Poměr signálu zpracovaného tímto efektem a čistého, neovlivněného signálu je možné nastavit pomocí parametru *Dry/Wet*.

Modul *Output* (13) slouží k úpravě celkové výstupní úrovně, čímž je možné omezit zkreslení vznikající při generování signálu více hlasy současně.

K ovládání nástroje lze použít také virtuální klaviaturu ve spodní části aplikace, která reaguje na akce kurzoru myši (14).

B Seznam tříd

Tato příloha poskytuje kompletní výčet tříd vytvořených v rámci této práce. Pokud není uvedeno jinak, daná třída odpovídá jednomu, stejně nazvanému, zdrojovému souboru. Detailnější popis tříd, jejich metod a datových členů je k dispozici v dokumentaci, která je součástí zdrojového kódu.

B.1 Globální pomocné datové struktury

`AdditiveGlobal`
: uchovává základní údaje o syntezátoru vyžadované třídami pro grafickou i výpočetní část.

`AdditiveParameters`
: obsahuje statickou funkci, která vytvoří parametry zásuvného modulu.

`ParameterIdentifiers`
: uchovává slovní identifikátory parametrů zásuvného modulu a poskytuje je ostatním třídám.

B.2 Výpočetní část

`PluginProcessor`
: uchovává všechny objekty pro zpracování signálu a vytváří třídu vykreslující grafické rozhraní.

`AdditiveAudioEngine`
: obsahuje instanci třídy Synthesiser a efektu reverb, kterým podle potřeby předává zvukový a MIDI data.

`AdditiveSynthesiser`
: viz kapitola 4.6.1

`AdditiveSynthesiserSound`
: viz kapitola 4.6.1

`AdditiveSynthesiserVoice`
: viz kapitola 4.6.1

`AdditiveOscillator`
: Implementuje generátory harmonických složek (viz kapitola 4.7).

B.3 Grafická část

`PluginEditor`
: obsahuje všechny instance tříd grafického rozhraní.

`StateReferencedComponent`
: tvoří bázovou třídu pro komponenty, které obsahují referenci na objekt `AudioProcessorValueTree`, ve které jsou uchovány parametry zásuvného modulu.

B.3.1 Grafické komponenty

`GlobalSettingsComponent`
: tvoří panel v horní části grafického rozhraní, ve kterém je vykreslen název aplikace.

Sekce nastavení harmonických složek

`OscillatorComponent`
: uchovává všechny komponenty určené pro ovládací prvky oscilátoru.

`SlidersComponent`
: vykresluje virtuální tahové potenciometry nastavující poměry harmonických složek.

`RightSideComponent`
: obsahuje tlačítka pro rychlé vyvolání speciálního poměru harmonických složek (např. obdélníkového nebo pilového signálu).

`LeftSideComponent`
: obsahuje virtuální otočné potenciometry, kterými lze vstupní MIDI noty posouvat o daný počet oktáv nebo půltónů.

`SpectrumMorphingComponent`
: obsahuje zmenšené náhledy na sady harmonických složek, mezi kterými signál prolíná.

Sekce pro ovládací prvky amplitudové obálky a efektů

`ControlsViewport`
: v grafickém rozhraní vytváří náhled na teoreticky neomezený prostor pro vykreslování grafických prvků, náhled je možné přesouvat pomocí posuvníků.

`ControlsComponent`
: uchovává všechny komponenty určené pro ovládání parametrů upravujících generovaný signál

`ADSRComponent`
: obsahuje virtuální otočné potenciometry pro ovládání parametrů amplitudové obálky.

`FilterComponent`
: obsahuje virtuální otočné potenciometry pro ovládání parametrů kmitočtového filtru.

`ReverbComponent`
: obsahuje virtuální otočné potenciometry pro ovládání parametrů efektu reverb.

`OutputComponent`
: obsahuje virtuální otočný potenciometr pro ovládání zesílení výstupního signálu.

B.3.2 Definice ovládacích prvků

`AdditiveLinearSlider`
: tvoří bázovou třídu pro definici vlastností a chování virtuálních tahových potenciometrů.

`AdditiveRotarySlider`
: tvoří bázovou třídu pro definici vlastností a chování virtuálních otočných potenciometrů.

`AdditiveTextButton`
: tvoří bázovou třídu pro definici vlastností a chování tlačítek obsahujících text.

B.3.3 Definice vzhledu komponent

`AdditiveColours`
: uchovává kódy barev používaných v grafickém rozhraní.

`AdditiveLookAndFeel`
: definuje vlastní vykreslování některých ovládacích prvků.

`NumericSliderLookAndFeel`
: definuje vykreslování posuvníku pro nastavení času prolínání mezi více stavy harmonických složek

`SliderWideLookAndFeel`
: definuje vykreslení širokého virtuálního tahového potenciometru pro nastavení poměru harmonických složek.

`SliderMidLookAndFeel`
: definuje vykreslení středně širokého virtuálního tahového potenciometru pro nastavení poměru harmonických složek.

`SliderThinLookAndFeel`
: definuje vykreslení úzkého virtuálního tahového potenciometru pro nastavení

poměru harmonických složek.