VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SOFTWAROVÝ MULTIEFEKT PRO POSTPRODUKCI POPULÁRNÍ HUDBY

SOFTWARE MULTI-EFFECT FOR POST-PRODUCTION OF POP MUSIC

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. TOMÁŠ TRKAL**
AUTHOR

VEDOUCÍ PRÁCE — **Doc. Dr. Ing. JAN ČERNOCKÝ**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Trkal Tomáš, Bc.**
Obor: Informační systémy
Téma: **Softwarový multiefekt pro postprodukci populární hudby**
Software Multi-Effect for Post-Production of Pop Music
Kategorie: Zpracování signálů

Pokyny:

1. Prostudujte základní i pokročilé nástroje a techniky využívané v profesionální postprodukci populární hudby.
2. Navrhněte komplexní softwarový systém, využitelný pro zpracování široké škály signálů, s důrazem na jednoduchost jeho použití.
3. Navržený systém implementuje jako VST modul a doplňte jej o vhodné monitory parametrů signálu (např. spektrum a hlasitost).
4. Sestavte reprezentativní množinu vstupních signálu a vytvořte pro ně presety.
5. Navrhněte, proveďte a vyhodnoťte uživatelské testy.
6. Vytvořte krátké video demonstrující Vaši práci.

Literatura:

- dle doporučení vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2, značná rozpracovanost bodu 3.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Černocký Jan, doc. Dr. Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato diplomová práce se zabývá návrhem a implementací komplexního softwarového systému určeného pro postprodukci populární hudby. Systém byl implementován jako zásuvný modul v programovacím jazyce C++ s využitím aplikačního frameworku JUCE, přičemž byl mimo jiné kladen důraz na vytvoření přehledného a intuitivního uživatelského rozhraní. Plugin disponuje sadou zvukových efektů a procesorů, jež mohou být uživatelem zapojeny do požadované grafové struktury. Pro méně zkušené uživatele je k dispozici databáze přednastavených presetů, které lze aplikovat na široké spektrum vstupních signálů.

Abstract

This diploma thesis deals with design and implementation of complex software system for post-production of popular music. The system was implemented as a plug-in module in C++ language using JUCE application framework. The emphasis was on creating a well arranged and intuitive graphic user interface. The plug-in provides a set of audio effects and processors that can be connected into the desired graph structure. For less experienced users, there is a database of preset configurations usable for a variety of input signals.

Klíčová slova

Postprodukce, mix, mastering, hudba, zvuk, signál, plugin, multiefekt, efekt, procesor, filtr, ekvalizér, zpožďovací linka, oscilátor, vibráto, flanger, chorus, dozvuk, kompresor, expander.

Keywords

Post-production, mix, mastering, music, sound, signal, plugin, multi-effect, effect, processor, filter, equalizer, delay line, oscillator, vibrato, flanger, chorus, reverb, compressor, expander.

Citace

TRKAL, Tomáš. *Softwarový multiefekt pro postprodukci populární hudby.* Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Černocký Jan.

Softwarový multiefekt pro postprodukci populární hudby

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Jana Černockého. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Trkal
21. května 2017

Poděkování

Chtěl bych poděkovat zejména Janu Černockému za vstřícnost při konzultacích, cenné rady a věcné připomínky, které vedly ke zkvalitnění této diplomové práce. Rád bych také poděkoval Lucii Sližové za profesionální pomoc při vytváření grafického uživatelského rozhraní, Ondřeji Žatkuliakovi, Zdenkovi Kamenickému a Václavu Štirskému za jejich drahocenný čas a sdílení dlouholetých zkušeností s postprodukcí hudby a všem, kteří se zúčastnili testování pluginu Ptolemaios a pomohli tak k jeho zdokonalení. Na závěr bych chtěl poděkoval rodině, přátelům a přítelkyni za pevné nervy a podporu během studia a vypracování práce.

Obsah

Seznam použitých zkratek

AAX	Avid Audio eXtension
APF	All-Pass Filter
ASIO	Audio Stream Input/Output
AU	Audio Units
BL	BLend
BPF	Band-Pass Filter
BPM	Beats Per Minute
BSF	Band-Stop Filter
BW	BandWidth
DAW	Digital Audio Workstation
DDL	Digital Delay Line
DFS	Depth-First Search
DSP	Digital Signal Processing
FB	FeedBack
FF	FeedForward
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
GUI	Graphical User Interface
HPF	High-Pass Filter
IIR	Infinite Impulse Response
LED	Light-Emitting Diode
LFO	Low-Frequency Oscillation
LPF	Low-Pass Filter
MIDI	Musical Instrument Digital Interface
RTAS	Real-Time AudioSuite
SDK	Software Development Kit
VST	Virtual Studio Technology
WTO	Wave Table Oscillator
XML	eXtensible Markup Language

Kapitola 1

Úvod

Pod pojmem postprodukce hudby se ukrývá proces, jehož primárním úkolem je sladit jednotlivé stopy hudební nahrávky v jeden kompaktní celek tak, aby každý nástroj v nahrávce nalezl své místo. Jinými slovy se jedná o úpravu zvuku hudebních nástrojů a poměrů mezi nimi. Postprodukce v sobě zahrnuje fázi střihu, mixu a masteringu a je v dnešní době jedním z klíčových faktorů, ovlivňujících úspěšnost skladby v komerčním prostředí. Pro detailnější popis konkrétních fází vzniku nahrávky si čtenáře dovolím odkázat na svoji bakalářskou práci [28], kde jsem se jimi podrobněji zabýval.

Cílem této diplomové práce je návrh a implementace komplexního softwarového systému určeného pro postprodukci populární hudby. Výsledný systém Ptolemaios má podobu zásuvného pluginu, který byl implementován v programovacím jazyce C++ s využitím aplikačního frameworku JUCE. Plugin disponuje sadou dílčích zvukových efektů a procesorů, jež lze zapojit do libovolné acyklické grafové struktury, čímž je uživateli umožněno sestavit si vlastní originální multiefekt. Pro méně zkušené uživatele je k dispozici databáze přednastavených presetů, jež mohou být využity jako výchozí bod pro úpravu široké škály vstupních signálů. Presety zahrnují nejen zapojení jednotlivých modulů, ale i detailní nastavení jejich parametrů. Plugin uživateli umožňuje i načítání a ukládání presetů vlastních. Grafické uživatelské rozhraní bylo navrženo a realizováno tak, aby jeho použití bylo co možná nejvíce intuitivní.

Text práce je členěn do několika samostatných kapitol. Na první úvodní kapitolu navazuje kapitola zabývající se průzkumem trhu; ta si za cíl klade nalezení vhodného směru, jimž by bylo možné se se při návrhu nového multiefektu ubírat. Samotným návrhem se zabývá kapitola 3. Podrobnému popisu digitálních algoritmů vybraných zvukových efektů a procesorů je věnována kapitola 4. Kapitola 5 pak poskytuje nezbytný formální základ teorie grafů, včetně popisu algoritmů pro detekci cyklu v orientovaném grafu. Zbývající kapitoly jsou zaměřeny na implementaci a testování pluginu Ptolemaios. Výsledky této diplomové práce jsou shrnuty v závěrečné kapitole.

Kapitola 2

Průzkum trhu

Tato úvodní kapitola se zabývá aktuální situací na poli komplexních softwarových zásuvných modulů, které se využívají v nahrávacích studiích pro moderní postprodukci populární hudby. Při výběru a následném hodnocení pluginů jsem bral v úvahu především jejich použitelnost, přehlednost, intuitivnost, vzhled, přítomnost přednastavení (presetů) a v neposlední řadě i jejich cenu. Cílem tohoto průzkumu je získat přehled o nynějších trendech v oblasti multiefektových pluginů a nalezení vhodného směru, jímž by bylo možné se při návrhu nového pluginu ubírat.

2.1 iZotope Alloy 2

Firma *iZotope*[1] patří dlouhodobě k celosvětové špičce mezi společnostmi, zabývajícími se právě tvorbou hudebního softwaru, zejména pak mixážních a masteringových multiefektů.

Obrázek 2.1: iZotope Alloy 2

[1] https://www.izotope.com

Následující text je věnován bližšímu seznámení s úspěšným mixážním multiefektem *iZotope Alloy 2* (obr. 2.1). Tento multiefekt disponuje ekvalizérem, transient shaperem[2], exciterem[3], dvěma dynamickými procesory, de-esserem a limiterem, což z něj činí mocný nástroj, využitelný pro úpravu širokého spektra zvukových signálů.

Jednotlivé moduly lze vypínat a zapínat a libovolně řadit do série. Nastavení parametrů modulů lze provádět ovládacími prvky, jež jsou umístěny na vlastních obrazovkách, které lze přepínat pomocí tlačítek nacházejících se v dolní části okna. Užitečnou novinkou oproti předchozí verzi pluginu je obrazovka *Overview*, která na jednom místě sdružuje klíčové ovládací prvky všech modulů.

Vzhled pluginu je přehledný, elegantní a koresponduje se vzhledem ostatních pluginů rodiny iZotope, což napomáhá uživateli se v celé sérii lépe orientovat. Méně zkušení uživatelé zajisté ocení velké množství přednastavených presetů, jež mohou využít jako výchozí bod své práce. Za zmínku také stojí kvalitní zpracování spektrálních analyzátorů a měřičů hlasitosti. Pokud bych měl tomuto multiefektu něco vytknout, byla by to absence zvukových efektů typu reverb, delay, flanger či chorus. Aktuální cena pluginu iZotope Alloy 2 se pohybuje okolo 150 $.

2.2 Waves GTR Stomp

Neméně známou firmou, která se zabývá vývojem audio efektů, je i společnost *Waves*[4]. Jedním z jejích úspěšných pluginů je i kytarový multiefekt *Waves GTR Stomp* (obr. 2.2). Tento plugin uživateli poskytuje šestadvacet kytarových procesorů, jejichž umístěním do prázdných slotů si lze poskládat požadovaný multiefekt, přičemž lze využít i zapojení více modulů stejného typu současně.

Obrázek 2.2: Waves GTR Stomp

[2] *Transient shaper* je dynamický procesor používaný pro zvýraznění či potlačení náběhu signálu.
[3] *Exciter* je zvukový procesor, který obohacuje signál o vyšší harmonické signály.
[4] http://www.waves.com

Jednotlivé moduly vzhledem připomínají hardwarové pedálové efekty, čemuž odpovídá i rozmístění ovládacích prvků, což umožňuje měnit parametry všech vybraných modulů pohodlně na jedné obrazovce. Na pravé straně obrazovky je umístěn diodový měřič výstupní hlasitosti, nicméně kvůli nepřítomnosti stupnice je jeho využití spíše orientační. K dispozici je široká škála přednastavení jak jednotlivých zvukových modulů, tak i jejich zapojení do slotů v rámci celého multiefektu. Waves GTR Stomp je samostatně neprodejný, ovšem lze jej zakoupit jako součást balíčků, jež lze pořídit za cenu od 129 $ až do 8000 $.

2.3 Positive Grid BIAS FX

Positive Grid[5] je firma zabývající se tvorbou a vývojem zejména analogových a digitálních kytarových efektů a procesorů. Mezi její aktuálně nejúspěšnější softwarové produkty se řadí dvojice multiefektů *BIAS Amp* a *BIAS FX*. Oba pluginy jsou si vizuálně velmi podobné, nicméně *BIAS FX* poskytuje rozšířenou funkcionalitu a výrazně vyšší počet modulů, což bylo i jedním z důvodů, proč jsem si pro bližší popis vybral právě tento plugin.

Obrázek 2.3: Positive Grid BIAS FX

Co se pluginu *BIAS FX* rozhodně nedá upřít, je jeho propracovaný a dle mého názoru i velmi zdařilý vzhled (obr. 2.3). Okno pluginu je rozděleno na tři části. První část obsahuje řetězec modulů, jejichž pořadí lze jednoduše měnit tahem myši. V základní verzi pluginu si uživatel může vybrat z dvanácti kytarových zesilovačů a třiceti efektů, přičemž profesionální verze pak tento výběr rozšiřuje na téměř dvojnásobek. Velmi zajímavou inovací je možnost rozdělení signálové cesty pomocí jednoho páru modulů *splitter* a *mixer* do dvou paralelních signálových cest.

[5] https://www.positivegrid.com

Po kliknutí na konkrétní modul se v druhé části obrazovky zobrazí detail vybraného efektu, který uživateli umožní měnit jeho parametry pomocí dostupných ovládacích prvků. Nutno podotknout, že základní verze pluginu neumožňuje automatizaci[6] těchto parametrů. Třetí a poslední část obrazovky poskytuje panel s ovládacími prvky pro změnu globálních parametrů a měřiči vstupní a výstupní hlasitosti signálu, které bohužel podobně jako u pluginu *Waves GTR Stomp* postrádají stupnice.

Obdobně jako předchozí dva multiefekty disponuje i *BIAS FX* bohatou bankou přednastavení zapojení modulů, která zahrnují i nastavení jejich parametrů. Cena základní verze pluginu *BIAS FX Desktop* je přibližně 100 $, verze *BIAS FX Professional* pak vychází zhruba na 200 $.

2.4 IK Multimedia T-RackS 3

Posledním multiefektem, který na závěr této kapitoly představím, je produkt původem italské vývojářské firmy *IK Multimedia*[7], jenž nese název *T-RackS* (obr. 2.4). Jedná se již o třetí řadu velmi úspěšného pluginu, jehož verze *Delux* v sobě ukrývá sadu šesti dynamických procesorů a tří ekvalizérů, které lze kombinovat libovolným zapojením do připravené rozdvojené signálové cesty čítající dvanáct volných slotů.

Obrázek 2.4: IK Multimedia TrackS 3

Přestože plugin *IK Multimedia T-RackS* nepatří mezi nejnovější produkty, zaujme bezpochyby uživatele svým nadčasovým vzhledem. Jednotlivé moduly nabízí pro ovládání svých parametrů ovládací prvky umístěné na vlastních obrazovkách připomínajících na první pohled reálná hardwarová zařízení. Tyto obrazovky můžeme přepínat tlačítky, která

[6] Automatizace je řízení parametru hostitelskou aplikací dle uživatelem nastavené automatizační křivky.
[7] http://www.ikmultimedia.com

se nacházejí v horní části okna pluginu. Z množiny všech dostupných parametrů jich lze vybrat maximálně šestnáct, jež mohou být plně automatizovány pomocí hostitelské aplikace. Tento počet sice není úplně ideální, nicméně pro většinu běžných uživatelů by měl být dostačující.

Na rozdíl od předcházejících dvou pluginů byl věnován výrazně větší prostor pro měřicí a analytické nástroje. Konkrétně se jedná o měřiče hlasitosti, fáze a spektrální analyzátor, jež můžete vidět na obrázku 2.4 v dolní části okna pluginu. Dále opět nalezneme nemalou množinu přednastavení parametrů zvukových procesorů včetně jejich zapojení do slotů. Plugin *IK Multimedia T-RackS* vyšel v několika verzích, které se od sebe navzájem liší především počtem a typem modulů a od toho odvozenou cenou; ta se pohybuje v rozmezí od 73 $ do zhruba 400 $.

2.5 Vyhodnocení průzkumu trhu

V této kapitole jsem se postupně věnoval analýze čtyř špičkových profesionálních multiefektů od čtyř různých celosvětově renomovaných vývojářských společností. U všech zmíněných produktů je kladen zvláštní důraz na přehlednou, propracovanou a uživatelsky přívětivou grafickou reprezentaci. Jednotnost vzhledu ať už jednotlivých částí nebo pluginu v rámci konkrétní série výrazně zvyšuje intuitivnost ovládání a tím pádem i použitelnost pluginu.

Kvalitnímu multiefektu by rozhodně neměla chybět zásoba přednastavení modulů využitelného pro široké spektrum vstupních zvukových signálů, protože zejména pro méně zkušené uživatele jsou často důležitým výchozím bodem při postprodukci jejich nahrávek. Stejně tak by měl dobrý plugin disponovat alespoň základními nástroji pro měření hlasitosti či analýzu signálu.

V návaznosti na tuto kapitolu jsem se rozhodl navrhnout a implementovat zvukový multiefekt v podobě zásuvného pluginu, který by na rozdíl od výše uvedených uživateli jednoduše umožnil zpracování signálu ve více paralelních větvích. Co se týká výběru vhodných zvukových algoritmů, nechtěl jsem se zaměřit na zpracování pouze jednoho typu signálu (například elektrických kytar), proto jsem vybral selekci různých zvukových efektů a procesorů, jež jsou včetně jejich principu podrobně probrány v kapitole 4.

Kapitola 3

Návrh aplikace

Tato kapitola se zabývá návrhem aplikace v podobě zásuvného modulu určeného pro zpracování zvukového signálu v reálném čase. Zásuvné moduly, neboli takzvané pluginy, běží v rámci hostitelské aplikace, která jim poskytuje audio data po částech prostřednictvím vyrovnávacích bufferů. Díky tomu může být zvukový signál upravován i během jeho přehrávání. Mezi zástupce hostitelských aplikací patří například *Image-Line Fruity Loops Studio*, *Apple Logic Pro* nebo již dříve zmíněné *Avid Pro Tools* a *Steinberg Cubase*.

3.1 Funkční požadavky

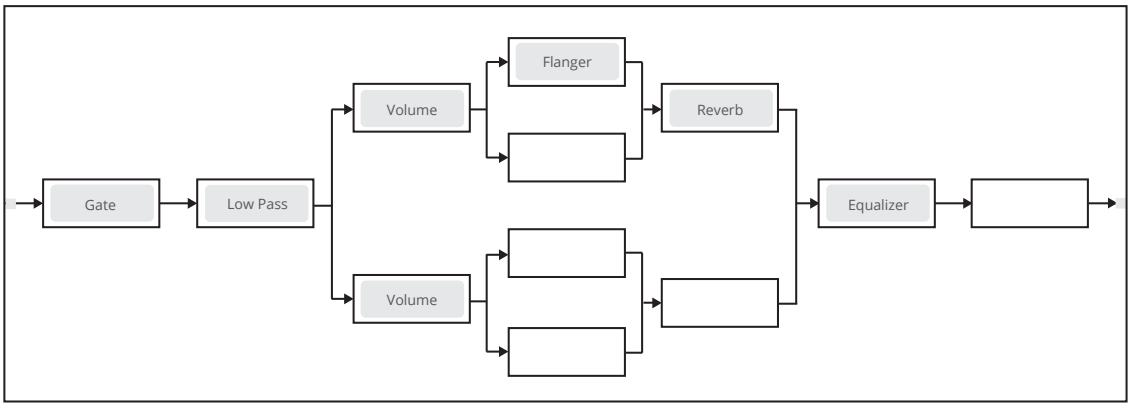
V kapitole 2 jsem uskutečnil průzkum trhu, jehož cílem bylo získat přehled o existujících profesionálních multiefektových pluginech. Na základě získaných znalostí a následujících konzultací s lidmi, kteří se postprodukcí populární hudby dlouhodobě zabývají, jsem sestavil seznam funkčních požadavků, jež jsem posléze využil pro návrh nového multiefektového pluginu. Hlavním směrem, kterým jsem se při návrhu rozhodl ubírat, *je možnost zpracování zvukového signálu ve více paralelních větvích*. Zmíněné funkční požadavky jsou uvedeny v následujícím výčtu:

- Sada zvukových efektů a procesorů v podobě modulů
- Dynamické přidávání a odebírání modulů za běhu.
- Možnost vytvoření více instancí jednoho typu modulu.
- Zpracování signálu ve více paralelních větvích.
- Monitory úrovně hlasitosti signálu.
- Grafická vizualizace nastavení některých parametrů.
- Automatizace parametrů hostitelskou aplikací.
- Sada přednastavených presetů.
- Možnost ukládání vlastních presetů.
- Jednoduché, přehledné a intuitivní ovládání.

3.2 Zpracování signálu ve více signálových větvích

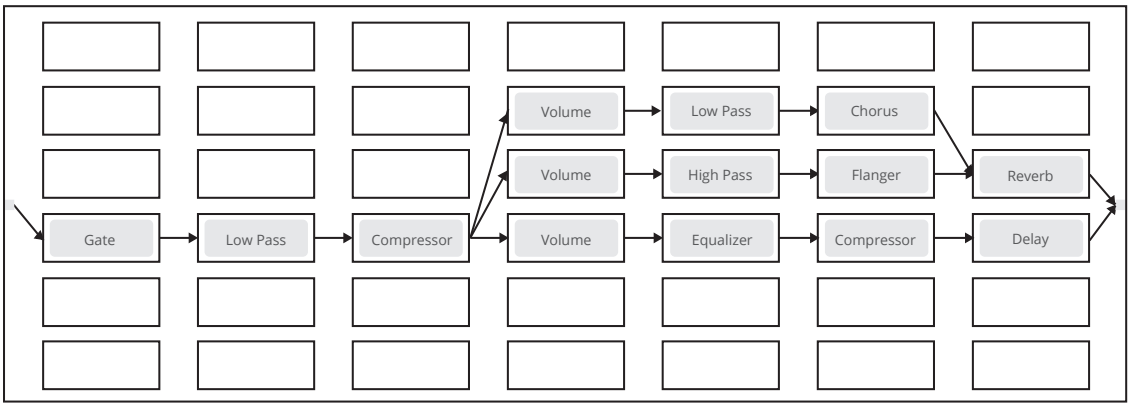
V této podkapitole jsem uvedl několik různých přístupů, kterými by bylo možné zpracování signálu ve více paralelních větvích v rámci nového pluginu zrealizovat. První a zároveň nejjednodušší varianta, vycházející z větvení signálu u pluginu *IK Multimedia T-RackS 3*, je zobrazena na obrázku 3.1.

Uživateli je zde poskytnuta sada slotů, které jsou propojeny do předem připravené struktury. Do těchto slotů pak uživatel může umisťovat instance jednotlivých zvukových efektů a procesorů. Výhodou tohoto řešení je snadná implementace, neboť pořadí procházení slotů při přehrávání je předem dáno. Nevýhodou je však absence možnosti přizpůsobit si počet větví či slotů vlastním potřebám. Tuto nevýhodu by bylo možné částečně kompenzovat poskytnutím více předpřipravených struktur propojených slotů, ze kterých by si uživatel mohl vybrat tu pro danou potřebu nejvíce vyhovující.



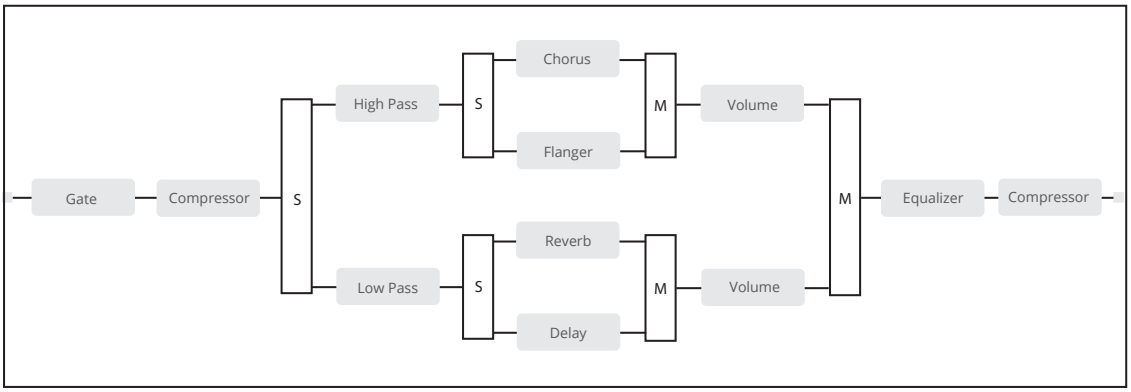
Obrázek 3.1: Návrh s připravenou strukturou slotů.

Na obrázku 3.2 je zobrazena druhá varianta, která disponuje mřížkou slotů, do nichž uživatel opět může vkládat instance zvukových efektů a procesorů. Na rozdíl od předchozí varianty však není propojení mezi sloty statické, nýbrž jej uživatel může dynamicky měnit za běhu. Propojit je vždy možné sloty pouze sousedních sloupců a to zleva doprava, přičemž počet vstupů a výstupů každého slotu je dán počtem slotů ve sloupci. Toto řešení umožňuje uživateli zapojit moduly do vlastních jednoduchých i komplexnějších struktur.



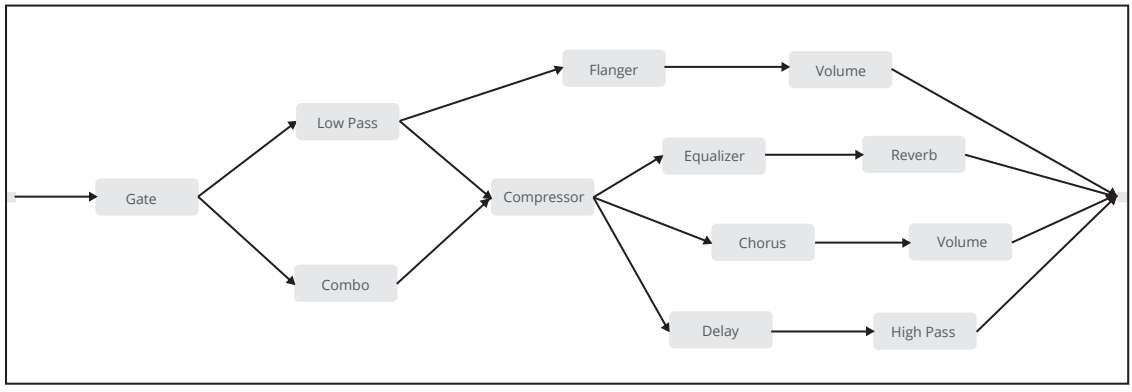
Obrázek 3.2: Návrh s mřížkou slotů a dynamickým propojováním.

Třetí varianta, jenž je zobrazena na obrázku 3.3, vychází ze způsobu paralelního větvení signálu u pluginu *Grid BIAS FX*. Pro vkládání modulů tato varianta oproti předchozím variantám nevyužívá slotů, ale umožňuje moduly umisťovat přímo na signálovou cestu. Tuto cestu je pak možné pomocí prvků *splitter* a *mixer* opakovaně větvit do dalších paralelních větví, nicméně vyšší stupeň zanoření může působit značně nepřehledně.



Obrázek 3.3: Návrh s větvením signálové cesty.

Čtvrtá a zároveň poslední varianta, která využívá větvení signálu podobně jako hostitelské aplikace *Propellerhead Reason* či *Juce Audio Plugin Host*, je zobrazena na obrázku 3.4. Tato varianta umožňuje uživateli za běhu vkládat instance zvukových efektů a procesorů a následně je propojovat do libovolné acyklické grafové struktury, přičemž počet vstupů a výstupů jednotlivých modulů není prakticky omezen. Výhodou této varianty oproti ostatním je vysoká variabilita zapojení, nevýhodou pak složitější implementace. Po každém přidání nebo odebrání modulu či propojení je zapotřebí vložené instance modulů topologicky přeuspořádat. K tomuto účelu jsem využil algoritmy uvedené v kapitole 5. Pro implementaci pluginu jsem si vybral právě tuto variantu.



Obrázek 3.4: Návrh s propojováním modulů do acyklické grafové struktury.

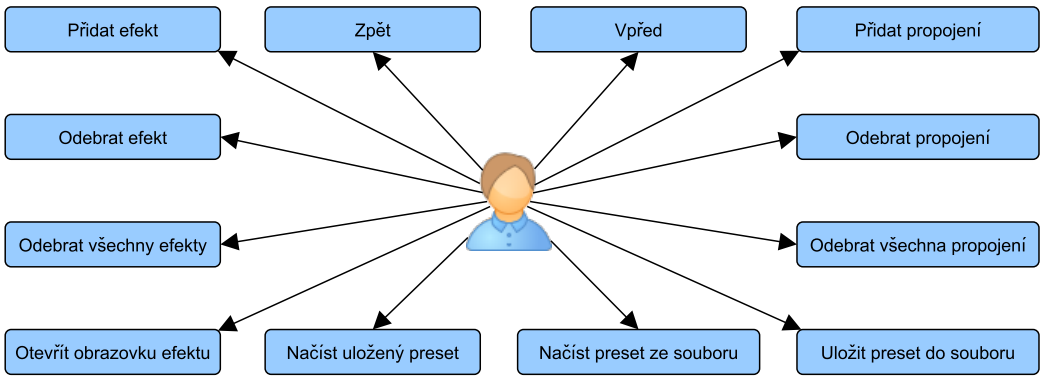
3.3 Návrh grafického uživatelského rozhraní

Nedílnou součástí každého pluginu je i jeho grafické uživatelské rozhraní (graphical user interface, GUI), jenž umožňuje uživateli plugin ovládat a které také často rozhoduje o obli-

benosti daného pluginu. Pokud plugin nedisponuje vlastním uživatelským rozhraním, je na základě jeho parametrů zpravidla generováno hostitelskou aplikací, což není zdaleka ideálním řešením, protože plugin pak vypadá v každé hostitelské aplikaci jinak, viz obrázek 3.5.

Obrázek 3.5: GUI automaticky generované různými hostitelskými aplikacemi.

Vzhled audio pluginů se na rozdíl od běžných aplikací velmi často neřídí aktuálními estetickými trendy z oblasti grafických uživatelských rozhraní. Nejčastěji se můžeme setkat s pluginy zastávající paradigma *skeumorfismu*, tedy s pluginy, jejichž vzhled imituje reálná studiová zařízení. Typickým příkladem takových pluginů jsou produkty firmy *Universal Audio*[1]. Na druhou stranu lze narazit i na velmi zdařilá vektorově založená uživatelská rozhraní. Zde jako vhodný příklad mohu uvést úspěšné produkty firmy *FabFilter*[2]. Osobně více preferuji první z těchto dvou variant, pročež jsem se touto cestou rozhodl jít i při tvorbě vlastního pluginu. Výčet základních operací grafického uživatelského rozhraní navrhovaného pluginu ilustruje obrázek 3.6.

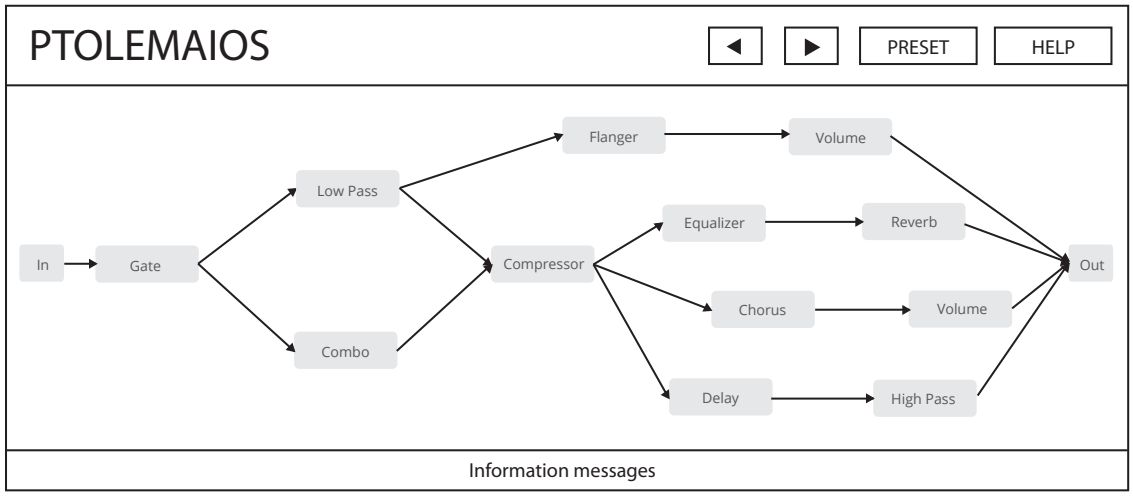


Obrázek 3.6: Výčet základních operací hlavní obrazovky pluginu.

Okno pluginu jsem rozdělil do těchto tří pod sebou ležících částí (obr. 3.7): *hlavní panel, ovládací panel a informační panel*. V levé části hlavního panelu je zobrazen název

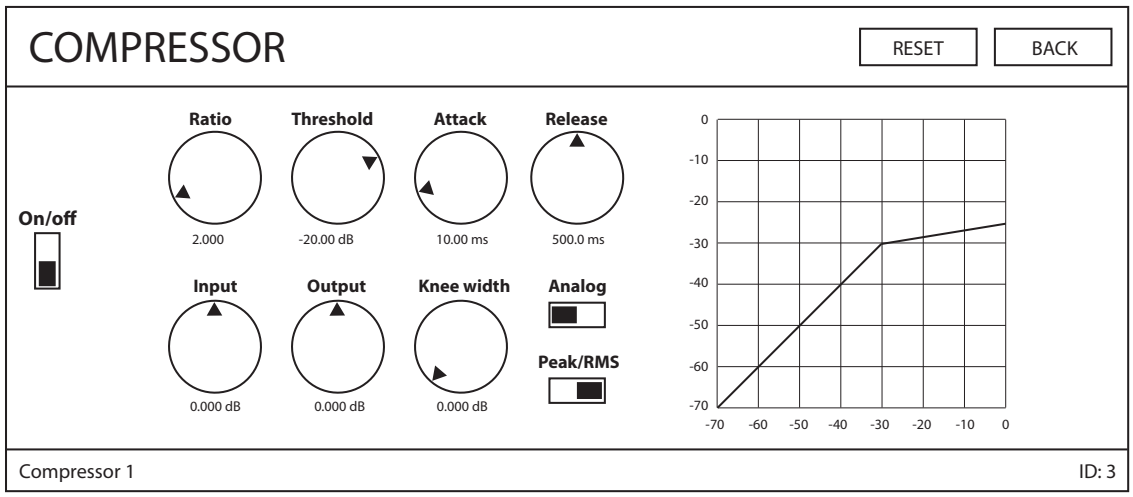
[1] http://www.uaudio.com
[2] http://www.fabfilter.com

pluginu, v pravé části je pak umístěna skupina čtyř tlačítek. První dvě tlačítka (*zpět* a *vpřed*) slouží k procházení historie změn. Tlačítko *preset* uživateli umožňuje načtení některého z připravených presetů, načtení presetu ze souboru či uložení aktuálního nastavení pluginu do souboru. Po kliknutí na tlačítko *help* si uživatel může zobrazit informace o pluginu, případně přejít na webové stránky, věnované využití pluginu.



Obrázek 3.7: Návrh hlavní obrazovky pluginu Ptolemaios.

Druhou částí okna pluginu je *ovládací panel*, pomocí něhož uživatel může vytvářet vlastní zapojení modulů. V předchozí sekci 3.2 jsem podrobněji popsal čtyři různé varianty, jimiž by bylo možné realizovat zpracování signálu ve více paralelních větvích. Z těchto variant jsem si při návrhu pluginu vybral právě tu poslední. Výběr modulů bude realizován prostřednictvím kontextové nabídky; ta se zobrazí po kliknutí pravým tlačítkem myši do oblasti ovládacího panelu. Pomocí této nabídky bude také možné odstranit všechny vložené moduly, případně všechna jejich propojení. Vložené moduly bude možné propojit tažením myši. Poslední částí okna pluginu je *informační panel*, který, jak už jeho název napovídá, slouží k výpisu informačních a chybových zpráv.



Obrázek 3.8: Návrh ovládací obrazovky modulu kompresoru.

Ovládání jednotlivých modulů jsem se rozhodl separovat do vlastních obrazovek, jež po dvojkliku na konkrétní vložený modul překryjí hlavní obrazovku pluginu. Po otevření ovládací obrazovky vybraného modulu je na hlavním panelu zobrazen jeho název společně s tlačítky *reset* a *back*. Tlačítko *reset* může uživatel použít pro obnovení výchozího nastavení parametrů modulu, tlačítko *back* pak pro návrat na hlavní obrazovku pluginu. Ovládací panel obsahuje ovládací a vizualizační prvky, jež jsou podrobněji popsány v následujících dvou sekcích. Na informačním panelu je pak zobrazena identifikace právě otevřeného modulu.

3.3.1 Ovládací prvky

Digitální zvukové efekty a procesory využívají pro ovládání svých parametrů několika typů ovládacích prvků, jež jsou charakteristické právě pro audio aplikace, protože vznikly jako digitální reprezentace ovládacích prvků fyzických studiových zařízení. Typickým ovládacím prvkem, jenž se používá například pro řízení hlasitosti jednotlivých kanálů mixážního pultu, je *tahový potenciometr*, známý též pod názvem *fader*. U efektů se častěji než s tahovým potenciometrem můžeme setkat s *potenciometrem otočným* neboli *knobem*. Většina profesionálních pluginů umožňuje změnu daného parametru provést kromě tažením jezdce faderu či otočením knobu také zadáním konkrétní hodnoty do příslušného textového pole.

Obrázek 3.9: Ukázka různých typů tahových a otočných potenciometrů.

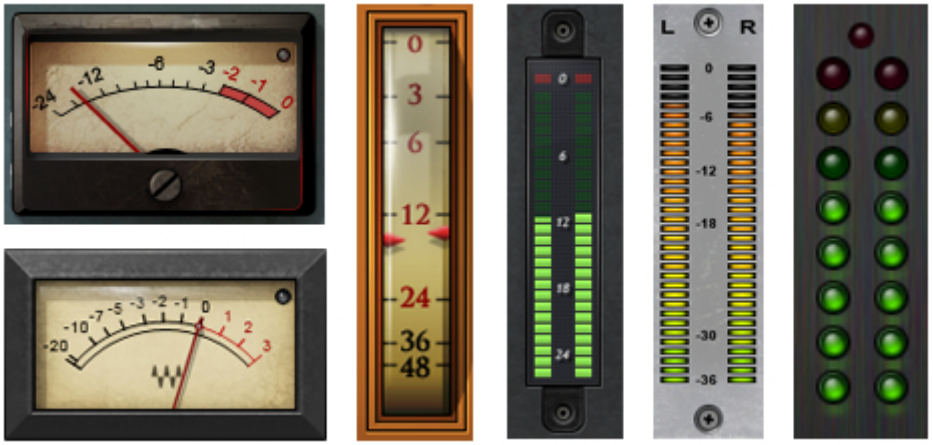
Kromě parametrů, jejichž hodnota se pohybuje v určitém intervalu, můžeme narazit i na parametry stavové. Pro změnu parametru se dvěma stavy se zpravidla využívá tlačítka v podobě přepínače. Pro změnu parametrů s více než dvěma stavy (např. typ filtru) není již využití přepínačů praktické, pročež se využívá rozbalovacího seznamu (combo box).

3.3.2 Vizualizační prvky

Vizualizační prvky grafického uživatelského rozhraní audio aplikací pomáhají uživateli analyzovat zpracovávaný zvukový signál případně kontrolovat nastavení parametrů. Nejběžnějším vizualizačním prvkem, který nalezneme takřka u každého profesionálního pluginu je indikátor hlasitosti signálu.

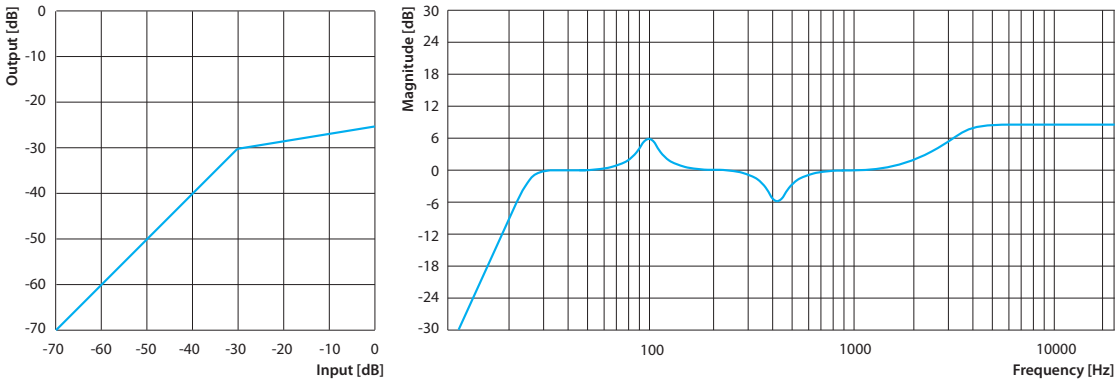
Vzhled měřičů hlasitosti se liší produkt od produktu. Indikátory hlasitosti některých pluginů připomínají reálné analogové měřící přístroje, u nichž je hlasitost signálu znázorněna

odpovídající výchylkou ručičky přístroje, jež ukazuje na danou hodnotu stupnice. Často se také můžeme setkat s indikátory využívající vertikálně či horizontálně orientovaného pásu světla vyzařujících diod (light-emitting diode, LED), které jsou v závislosti na hlasitosti signálu postupně rozsvěcovány.



Obrázek 3.10: Různé typy indikátorů hlasitosti signálu.

U některých zvukových efektů či procesorů je nastavení konkrétních parametrů pro lepší orientaci uživatele vhodné vizualizovat pomocí grafů. Ukázkou takového vizualizačního prvku je například graf zobrazující kompresní křivku, která pro dané nastavení parametrů kompresoru ukazuje závislost výstupní hlasitosti na hlasitosti vstupní (obr. 3.11 vlevo). Dalším u audio pluginů často vyskytujícím se vizualizačním prvkem je graf zobrazující ekvalizační křivku (obr. 3.11 vpravo); ta ukazuje zesílení, respektive zeslabení, jednotlivých frekvenčních složek signálu. Některé profesionální pluginy disponují takzvanými interaktivními grafy, které uživateli umožňují měnit tvar křivky, tedy i hodnoty parametrů, změnou polohy daných kontrolních bodů v rámci oblasti grafu.



Obrázek 3.11: Grafy zobrazující kompresní a ekvalizační křivku.

Kapitola 4

Digitální zpracování signálu

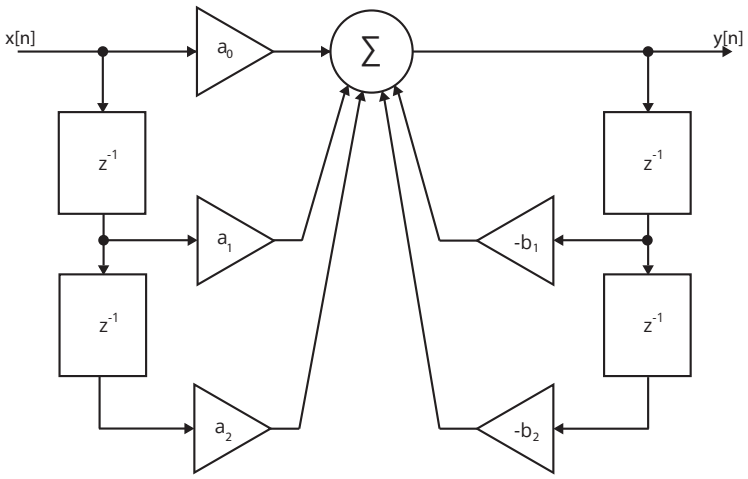
V této kapitole jsou popsány algoritmy digitálních zvukových procesorů a efektů, které jsem vybral na základě vlastních zkušeností, průzkumu trhu (viz. kapitola 2) a konzultací s lidmi, kteří se postprodukcí populární hudby profesionálně zabývají.

4.1 Filtry

Jako filtr lze obecně označit jakoukoliv komponentu, která z dané množiny vyselektuje určité prvky, jež splňují požadované vlastnosti, přičemž pokud tuto definici aplikujeme na digitální zpracování zvuku, provádí filtr výběr požadovaných složek signálu na základě jejich frekvence. Tyto složky pak můžeme použitím konkrétního typu filtru propouštět, potlačit či zvýraznit [32].

4.1.1 Biquad

Nejčastěji používanou strukturou digitálního filtru je takzvaný *biquad* [31]. Jedná se o filtr druhého řádu s nekonečnou impulsní odezvou, jehož blokový diagram je uveden na obrázku 4.1. Tato struktura umožňuje konkrétním nastavením koeficientů získat všechny běžné typy filtrů, jejichž výčet je uveden v následujícím textu.



Obrázek 4.1: Blokový diagram IIR filtru druhého řádu.

Koeficienty jsou vypočítány podle typu filtru, požadované frekvence f_c, vzorkovací frekvence f_s, zesílení *gain* a *faktoru Q (quality factor)* pomocí následujících rovnic:

$$\begin{aligned} V &= 10^{gain/20}, \\ K &= \tan(\pi f_c / f_s). \end{aligned} \tag{4.1}$$

Low-pass

a_0	a_1	a_2	b_1	b_2
$\frac{K^2}{1+K/Q+K^2}$	$\frac{2K^2}{1+K/Q+K^2}$	$\frac{K^2}{1+K/Q+K^2}$	$\frac{2(K^2-1)}{1+K/Q+K^2}$	$\frac{1-K/Q+K^2}{1+K/Q+K^2}$

High-pass

a_0	a_1	a_2	b_1	b_2
$\frac{1}{1+K/Q+K^2}$	$\frac{-2}{1+K/Q+K^2}$	$\frac{1}{1+K/Q+K^2}$	$\frac{2(K^2-1)}{1+K/Q+K^2}$	$\frac{1-K/Q+K^2}{1+K/Q+K^2}$

Band-pass

a_0	a_1	a_2	b_1	b_2
$\frac{K/Q}{1+K/Q+K^2}$	$\frac{0}{1+K/Q+K^2}$	$\frac{-K/Q}{1+K/Q+K^2}$	$\frac{2(K^2-1)}{1+K/Q+K^2}$	$\frac{1-K/Q+K^2}{1+K/Q+K^2}$

Band-stop

a_0	a_1	a_2	b_1	b_2
$\frac{1+K^2}{1+K/Q+K^2}$	$\frac{2(K^2-1)}{1+K/Q+K^2}$	$\frac{1+K^2}{1+K/Q+K^2}$	$\frac{2(K^2-1)}{1+K/Q+K^2}$	$\frac{1-K/Q+K^2}{1+K/Q+K^2}$

Peak (boost)

a_0	a_1	a_2	b_1	b_2
$\frac{1+VK/Q+K^2}{1+K/Q+K^2}$	$\frac{2(K^2-1)}{1+K/Q+K^2}$	$\frac{1-VK/Q+K^2}{1+K/Q+K^2}$	$\frac{2(K^2-1)}{1+K/Q+K^2}$	$\frac{1-K/Q+K^2}{1+K/Q+K^2}$

Peak (cut)

a_0	a_1	a_2	b_1	b_2
$\frac{1+K/Q+K^2}{1+VK/Q+K^2}$	$\frac{2(K^2-1)}{1+VK/Q+K^2}$	$\frac{1-K/Q+K^2}{1+VK/Q+K^2}$	$\frac{2(K^2-1)}{1+VK/Q+K^2}$	$\frac{1-VK/Q+K^2}{1+VK/Q+K^2}$

Low-shelving (boost)

a_0	a_1	a_2	b_1	b_2
$\frac{1+\sqrt{2V}K+VK^2}{1+\sqrt{2}K+K^2}$	$\frac{2(VK^2-1)}{1+\sqrt{2}K+K^2}$	$\frac{1-\sqrt{2V}K+VK^2}{1+\sqrt{2}K+K^2}$	$\frac{2(K^2-1)}{1+\sqrt{2}K+K^2}$	$\frac{1-\sqrt{2}K+K^2}{1+\sqrt{2}K+K^2}$

Low-shelving (cut)

a_0	a_1	a_2	b_1	b_2
$\frac{1+\sqrt{2}K+K^2}{1+\sqrt{2V}K+VK^2}$	$\frac{2(K^2-1)}{1+\sqrt{2V}K+VK^2}$	$\frac{1-\sqrt{2}K+K^2}{1+\sqrt{2V}K+VK^2}$	$\frac{2(VK^2-1)}{1+\sqrt{2V}K+VK^2}$	$\frac{1-\sqrt{2V}K+VK^2}{1+\sqrt{2V}K+VK^2}$

High-shelving (boost)

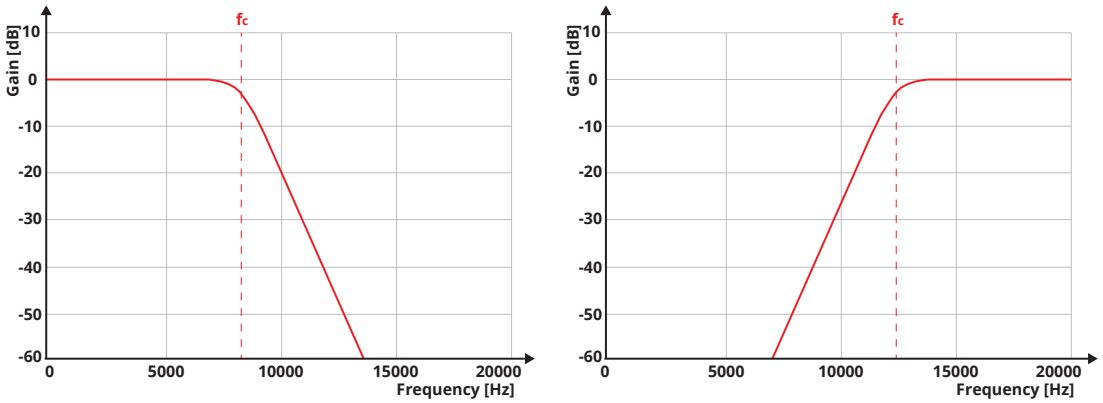
a_0	a_1	a_2	b_1	b_2
$\frac{V+\sqrt{2V}K+K^2}{1+\sqrt{2}K+K^2}$	$\frac{2(K^2-V)}{1+\sqrt{2}K+K^2}$	$\frac{V-\sqrt{2V}K+K^2}{1+\sqrt{2}K+K^2}$	$\frac{2(K^2-1)}{1+\sqrt{2}K+K^2}$	$\frac{1-\sqrt{2}K+K^2}{1+\sqrt{2}K+K^2}$

High-shelving (cut)

a_0	a_1	a_2	b_1	b_2
$\frac{1+\sqrt{2}K+K^2}{V+\sqrt{2V}K+K^2}$	$\frac{2(K^2-1)}{V+\sqrt{2V}K+K^2}$	$\frac{1-\sqrt{2}K+K^2}{V+\sqrt{2V}K+K^2}$	$\frac{2(VK^2-1)}{V+\sqrt{2V}K+K^2}$	$\frac{1-\sqrt{2V}K+VK^2}{V+\sqrt{2V}K+K^2}$

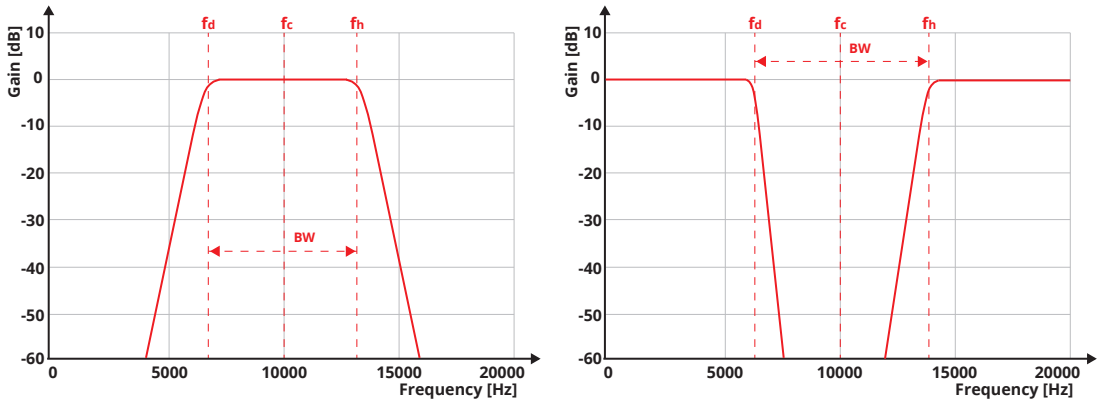
Tabulka 4.1: Koeficienty pro různé typy filtrů [16].

Highpass neboli *horní propust* patří společně s dolní propustí mezi nejpoužívanější typy filtrů. Jak už název napovídá, propouští tento typ filtru pouze vyšší frekvence, přičemž naopak nižší frekvence jsou výrazně potlačeny. Hranici mezi propouštěnými a nepropouštěnými frekvencemi lze ovlivnit pomocí změny parametru mezní frekvence f_c *(corner frequency)*. Highpass filtry jsou hojně využívány při postprodukci hudby zejména pro eliminaci hluků ve spodním pásmu frekvenčního spektra.



Obrázek 4.2: Frekvenční odezva lowpass a highpass filtru.

Lowpass neboli *dolní propust* propouští analogicky k předchozímu typu filtru výhradně frekvence, které jsou nižší než mezní frekvence f_c a potlačuje vyšší. Některé lowpass a highpass filtry disponují parametrem *Q faktor (quality factor)*, díky němuž mohou na mezní frekvenci vytvořit rezonanci. Lowpass filtry lze využít například v kombinaci s automatizací pro navození efektu potápění se pod hladinu.



Obrázek 4.3: Frekvenční odezva bandpass a bandstop filtru.

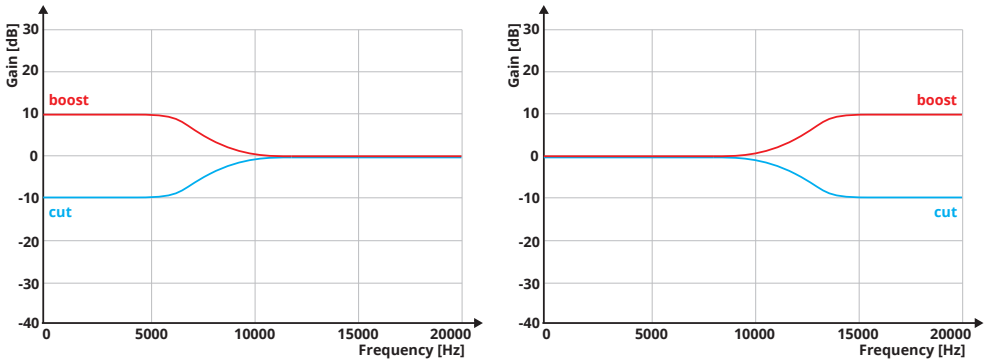
Bandpass neboli *pásmová propust* je typem filtru, jenž propouští frekvenční pásmo signálu, které je zdola ohraničeno mezní frekvencí f_d a shora mezní frekvencí f_h. Rozdíl těchto dvou frekvencí nazýváme *šířkou pásma (bandwidht, BW)*. Někdy bývá pásmová propust alternativně popsána právě šířkou pásma a střední frekvencí f_c. Díky pásmové propusti je například možné omezit frekvenční rozsah zvukové stopy a docílit tak telefonního efektu.

Bandstop neboli *pásmová zádrž* naopak tlumí určité frekvenční pásmo, které je opět definováno buďto mezními frekvencemi f_d a f_h nebo šířkou pásma BW a střední frekvencí f_c. Pásmová zádrž s velmi úzkou šířkou pásma se nazývá *vrubový filtr (notch)*.

Peak je typ filtru definovaný střední frekvencí f_c, faktorem Q a zesílením *gain* umožňující potlačit nebo naopak zvýraznit část frekvenčního spektra signálu.

Highshelf je typ filtru, který umožňuje potlačit či zvýraznit vyšší frekvenční pásmo signálu, jenž je zespodu ohraničeno parametrem mezní frekvence f_c. Míru zeslabení, respektive zesílení, tohoto pásma lze ovlivnit změnou parametru *gain*.

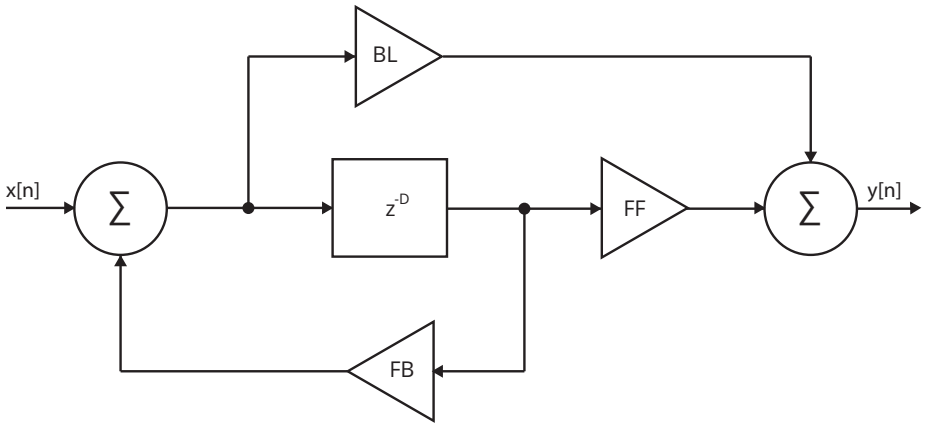
Lowshelf je obdoba filtru highself pro nízké frekvenční pásmo.



Obrázek 4.4: Frekvenční odezva lowshelf a highself filtru.

4.1.2 Univerzální hřebenový filtr

Mezi nejznámější struktury digitálních filtrů lze bezesporu zařadit i strukturu takzvaného *univerzálního hřebenového filtru* (obr. 4.5). Tento filtr disponuje třemi parametry: *blend (BL)*, *feedback (FB)* a *feedforward (FF)*. Změnou těchto parametrů lze snadno vytvořit několik jednoduchých efektů, jež velmi často nacházejí uplatnění jako dílčí komponenty složitějších zvukových systémů. Nastavení parametrů určující konkrétní typ efektu jsou uvedena v tabulce 4.2 [32].



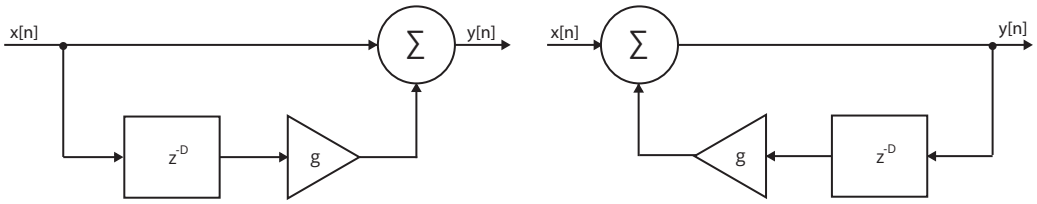
Obrázek 4.5: Blokový diagram univerzálního hřebenového filtru [32].

	BL	FB	FF
FIR hřebenový filter	1	0	g
IIR hřebenový filter	1	g	0
fázovací článek	a	-a	1
zpožďovací linka	0	0	1

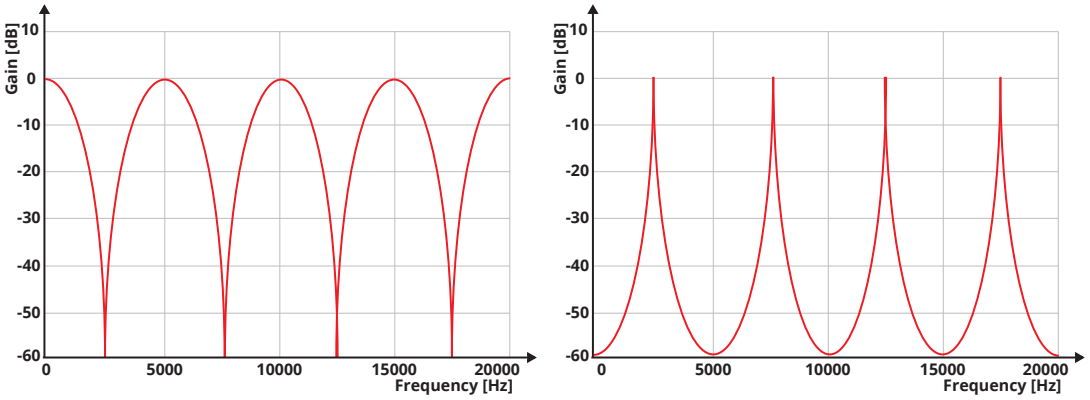
Tabulka 4.2: Parametry pro univerzální hřebenový filtr [32].

Hřebenový filtr

Hřebenový filtr (comb filter) je základním stavebním kamenem mnoha zvukových efektů. Název si tento typ filtru získal podle své frekvenční charakteristiky, jenž tvarem připomíná hřeben. Pomocí struktury univerzálního hřebenového filtru lze definovat dvě varianty hřebenového filtru: *dopřednou (feedforward)* a *zpětnovazební (feedback)* [23].



Obrázek 4.6: Blokové schéma FIR a IIR hřebenového filtru.



Obrázek 4.7: Frekvenční odezva FIR a IIR hřebenového filtru.

Fázovací článek

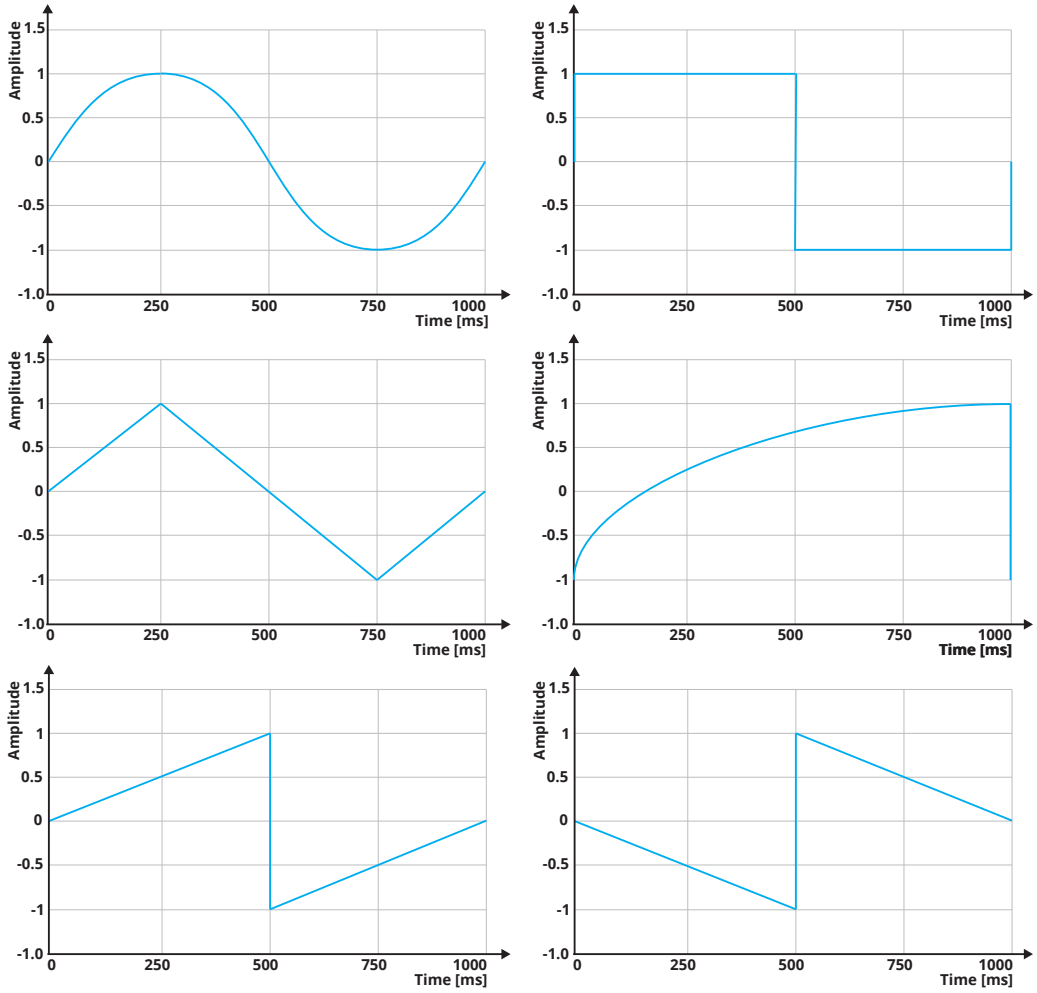
Fázovací článek (all-pass filter) je speciální typ filtru, jenž opět vychází ze struktury univerzálního hřebenového filtru. Oproti ostatním filtrům propouští fázovací článek všechny frekvence, přičemž však ale mění jejich fázi. Tento filtr zpravidla disponuje jediným parametrem, kterým je střední frekvence f_c. Vezmeme-li v úvahu fázovací článek prvního řádu, pak f_c značí frekvenci, jejíž fáze byla posunuta o 90°. V případě fázovacího článku druhého řádu by byla posunuta o 180°. Fázovací články nachází uplatnění například jako součást phaserů či reverbů [32].

4.2 Frekvenční oscilátory

Jako oscilátor lze označit zařízení, jenž samo o sobě nezpracovává ani nepřijímá žádný vstupní signál, ale naopak zdrojem signálu je. V odvětví analogové studiové techniky i digitálního zpracování signálu nacházejí frekvenční oscilátory dlouhodobě široké uplatnění. Kromě sinusového signálu mohou oscilátory běžně produkovat také obdélníkový, trojúhelníkový, exponenciální či pilový signál, viz obrázek 4.8.

Nejčastěji se s frekvenčním oscilátorem můžeme setkat jako se součástí syntetizérů, což jsou analogové či digitální hudební nástroje, které svůj výstupní zvuk produkují pomocí aditivní syntézy dílčích jednoduchých signálů, jež jsou generovány právě frekvenčními oscilátory. Druhým neméně významným využitím je řízení parametrů jiného efektu prostřednictvím oscilátorem generovaného signálu, jehož frekvence zpravidla nepřesahuje 20 Hz. Z tohoto důvodu bývají tyto oscilátory označovány jako *nízkofrekvenční (low-frequency oscillator, LFO)* [17].

V praxi se pro modulaci parametrů například zpožďovacích efektů využívají zejména signály sinusového či trojúhelníkového průběhu, protože skoková změna amplitudy ostatních uvedených signálů způsobuje nežádoucí zvukové artefakty, které se negativně podepisují na kvalitě výsledku [3].



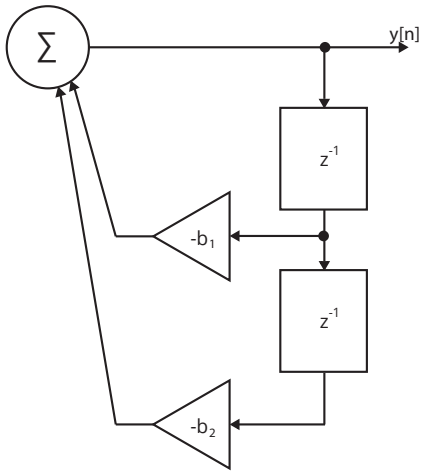
Obrázek 4.8: Příklady jedné periody různých signálů o frekvenci 1 Hz.

4.2.1 Přímá forma oscilátoru

Přímá forma je jedním ze základních způsobů implementace digitálního oscilátoru, která vychází ze struktury IIR filtru druhého řádu, jenž byla zmíněna v kapitole 4.1. Jak lze vidět na obrázku 4.9, tato forma oscilátoru využívá pouze rekurzivní části filtru. V komplexní rovině je tento oscilátor reprezentován dvojicí komplexně sdružených pólů ležících přesně na jednotkové kružnici. Koeficienty filtru lze vypočítat pomocí:

$$\begin{aligned} b_1 &= -2\cos\Omega, \\ b_2 &= 1, \\ \Omega &= 2\pi f_t / f_s, \end{aligned} \tag{4.2}$$

kde f_t je požadovaná frekvence oscilátoru a f_s vzorkovací frekvence. Před spuštěním je nezbytné vypočítat počáteční hodnoty uložené ve zpožďovacích blocích. Tento typ oscilátoru bohužel neumožňuje současně s generovaným signálem produkovat i signál o 90° fázově posunutý. K tomuto účelu lze využít například *Gordon-Smithův oscilátor* [4].



Obrázek 4.9: Blokové schéma přímé formy oscilátoru [14].

4.2.2 Tabulkový oscilátor

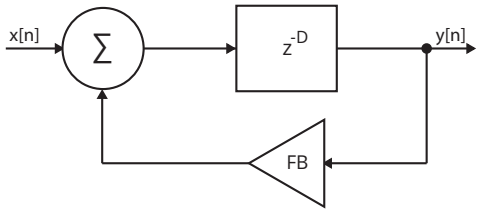
Druhá možnost implementace je založena na využití tabulky ve formě kruhového bufferu, ve které je uložena jedna perioda signálu požadovaného průběhu. Frekvence tohoto signálu je určena podílem vzorkovací frekvence a délky bufferu. Pro získání signálu o libovolné frekvenci je nutné vypočítat velikost kroku pomocí vztahu:

$$step = L\frac{f_t}{f_s}, \tag{4.3}$$

kde L je délka bufferu, f_t požadovaná frekvence signálu a f_s vzorkovací frekvence. Nutno podotknout, že velikost kroku není zpravidla celé číslo, proto je nezbytné pro získání hodnoty ležící mezi dvěma uloženými vzorky použít lineární nebo polynomiální interpolaci. Výhodou tohoto typu oscilátoru je nízká časová složitost a také možnost generování celé řady průběhů signálu [15].

4.3 Zpožďovací linka

Zpožďovací linka (digital delay line, DDL) je základním stavebním kamenem řady zvukových efektů, z nichž ty nejpoužívanější jsou popsány v textu následujících podkapitol. Jak už sám název napovídá, tento prvek umožňuje zpožďovat vstupní signál o určitou dobu. Před nástupem číslicových systémů bývaly k tomuto účelu většinou využívány kotoučové magnetofony s jednou záznamovou hlavou a jednou či více hlavami čtecími. S postupem času však byly tyto přístroje prakticky nahrazeny právě pomocí digitálních technologií. Struktura digitální zpožďovací linky vychází z architektury univerzálního hřebenového filtru, která je uvedena na obrázku 4.5.



Obrázek 4.10: Jednoduchá zpožďovací linka se zpětnou vazbou.

Digitální zpožďovací linka bývá zpravidla implementována formou kruhového bufferu, přičemž zápis a čtení vzorků probíhá prostřednictvím zápisového a čtecího ukazatele, jež odkazují na pozice v bufferu, které jsou od sebe vzdáleny požadovaný počet vzorků. Nutno podotknout, že čtení uloženého vzorku je vždy provedeno před zápisem vzorku nového, což je důležité v situacích, kdy čtecí i zápisový ukazatel odkazují na totožnou pozici. Po každé iteraci dochází k inkrementaci obou ukazatelů, načež je v případě přetečení hranice bufferu postižený ukazatel opět nastaven na počáteční pozici bufferu. Podobně jako u jejich analogových předchůdců se můžeme setkat s variantou zpožďovací linky, která disponuje více čtecími ukazateli [14].

Efekt	**Delay [ms]**	**Feedback [%]**	**Wet [%]**	**Dry [%]**	**Modulation [Hz]**
Delay	> 25	0 - 100	0 - 100	0 - 100	žádná
Slapback	60 - 150	0	0 - 100	0 - 100	žádná
Vibráto	0 - 10	0	100	0	0.1 - 5
Flanger	0 - 10	-100 - 100	50	50	0.1 - 1
Chorus	7 - 40	typicky 0	50	50	0.02 - 5

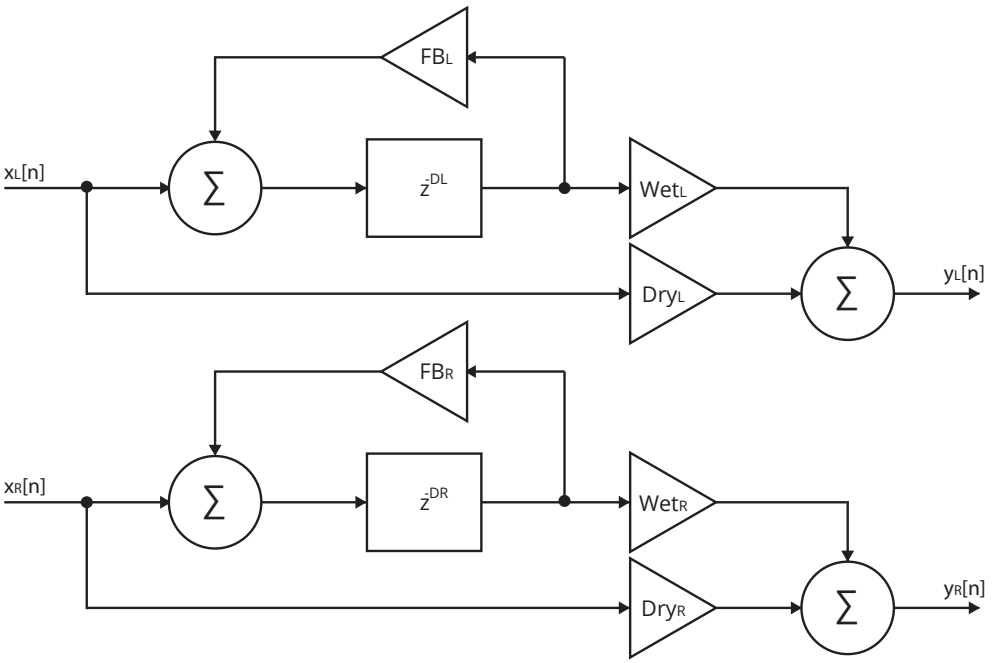
Tabulka 4.3: Nastavení parametrů zpožďovací linky pro konkrétní efekty [14].

V tabulce 4.3 je uveden výčet zvukových efektů, které lze získat s využitím zpožďovací linky vhodným nastavením rozsahů jednotlivých parametrů. Základním parametrem zpožďovací linky je *délka zpoždění (delay)*. Parametr *dry* ovlivňuje velikost přímého signálu, parametr *wet* velikost signálu zpožděného a parametr *feedback* velikost zpětné vazby. Často se zpožďovací linka používá v kombinaci s nízkofrekvenčním oscilátorem, jehož úkolem je periodicky měnit dobu zpoždění signálu. Zde je nutné si uvědomit, že přesnost zpoždění na jeden vzorek není dostačující, proto je nutné pro výpočet hodnoty mezi dvěma sousedními vzorky využít přinejmenším lineární interpolace. V opačném případě totiž hrozí výskyt nežádoucích zvukových artefaktů [32].

4.3.1 Delay

Prvním a zároveň nejjednodušším efektem založeným na zpožďovací lince se zpětnou vazbou je *delay*. Jedná se o efekt, který na výstupu kombinuje přímý signál se signálem zpožděným, čímž vytváří ozvěnu. Aby bylo možné od sebe oba signály rozpoznat, musí být doba zpoždění minimálně 25 ms [19]. Poměr těchto dvou signálů může uživatel ovlivnit pomocí parametrů *wet* a *dry*. Parametr *feedback* pak umožňuje řídit velikost zpětné vazby a tím korigovat počet opakování signálu.

Nejdůležitějším parametrem je pravděpodobně samotná doba zpoždění signálu, kterou lze zpravidla nastavit dvěma různými způsoby. Prvním z nich je zadání konkrétní doby zpoždění, přičemž rozsah zpoždění se liší v závislosti na konkrétním pluginu. Obvykle se však můžeme setkat s rozsahem pohybujícím se od jednotek milisekund až po jednotky sekund. Druhou a velmi užitečnou možností je synchronizace doby zpoždění dle násobků délky čtvrťové noty, která je vypočítána na základě aktuálního *tempa skladby (beats per minute, BPM)*. Často se dokonce můžeme setkat s možností odvodit zpoždění signálu i dle násobků délky čtvrťových not triolových či tečkovaných.



Obrázek 4.11: Blokový diagram stereo delay.

Na obrázku 4.11 je zobrazen blokový diagram stereofonní varianty efektu delay, která disponuje dvěma zpožďovacími linkami se zpětnou vazbou, což umožňuje nastavení zpoždění signálu pro každý kanál samostatně. Pokud se délka zpoždění pohybuje mezi 60 a 150 ms a zpětná vazba nepropouští signál, bývá delay označován přívlastkem *slapback*. Kromě výše uvedené struktury se můžeme setkat s celou řadou modifikací. Jako příklad lze uvést *multi-tap* či *ping-pong delay*. I přes svoji jednoduchost je delay velmi oblíbeným a hojně používaným zvukovým efektem, který lze v praxi aplikovat na téměř libovolný hudební nástroj či zpěv [30].

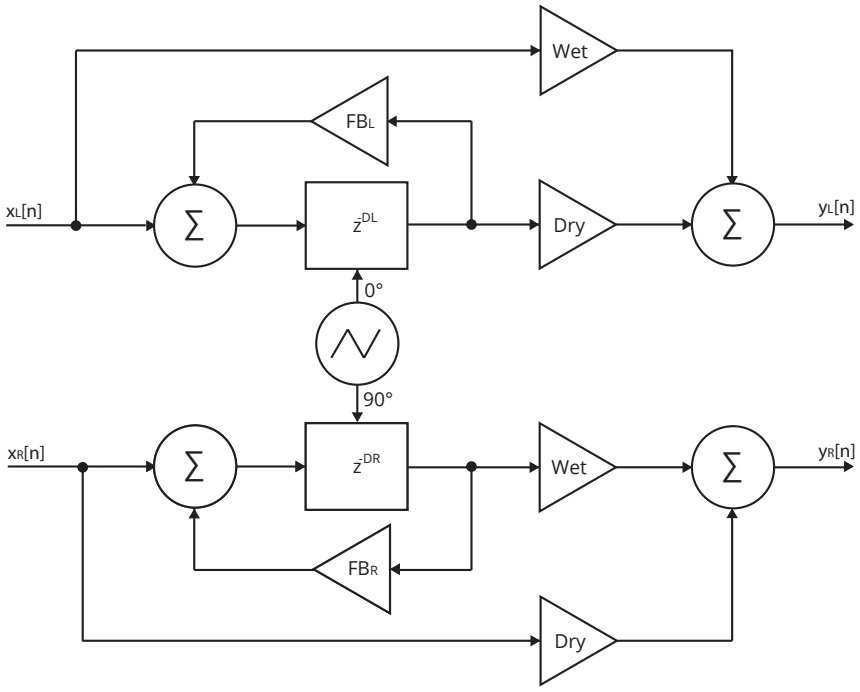
4.3.2 Vibráto

Vibráto je zvukový efekt, který je založený na zpožďovací lince s proměnnou délkou zpoždění řízenou nízkofrekvenčním oscilátorem. Tento efekt napodobuje pokročilou hudební techniku využívanou zpěváky a muzikanty, při niž dochází k drobným, rychlým a periodickým výchylkám tónu hudebního nástroje či zpěvu [17]. Klasickým příkladem této techniky je hra na housle, při níž hudebník rychlými dopřednými a zpětnými pohyby prstu po hmatníku ovlivňuje délku struny a tím i výšku tónu. Vibráto bývá často nesprávně zaměňováno za efekt *tremolo*, který však nemění periodicky výšku tónu nýbrž jeho hlasitost.

Oproti ostatním efektům založeným na zpožďovací lince je výstupem vibráta pouze zpožděný signál bez přidaného původního signálu [14]. Další odlišností zpravidla bývá absence zpětné vazby. Frekvenci oscilátoru může uživatel ovlivnit pomocí parametru *rate*, přičemž některá vibráta umožňuji tuto frekvenci synchronizovat s tempem hudební skladby. Zpoždění signálu se u vibráta pohybuje v rozpětí 0 až 7 ms, avšak tento rozsah lze omezit změnou parametru *depth*. Pro modulaci je většinou využíván sinusový nebo trojúhelníkový signál. Na použití pilového či obdélníkového signálu lze narazit zřídka, neboť oba produkují nežádoucí zvukové artefakty.

4.3.3 Flanger

Dalším běžně využívaným efektem, jenž je založen na zpožďovací lince s proměnnou délkou zpoždění, je *flanger*. Historie flangeru sahá až do začátku druhé poloviny minulého století, kdy tohoto efektu bylo dosahováno pomocí dvou paralelně zapojených kotoučových magnetofonů, jenž současně přehrávaly identický signál, přičemž vždy jeden z magnetofonů periodicky jemně snižoval a zvyšoval rychlost přehrávání [17]. Charakteristického zvuku flangeru, připomínající víření, bylo tedy docíleno vlivem konstruktivní a destruktivní interference těchto dvou signálů.



Obrázek 4.12: Blokový diagram stereo flangeru [14].

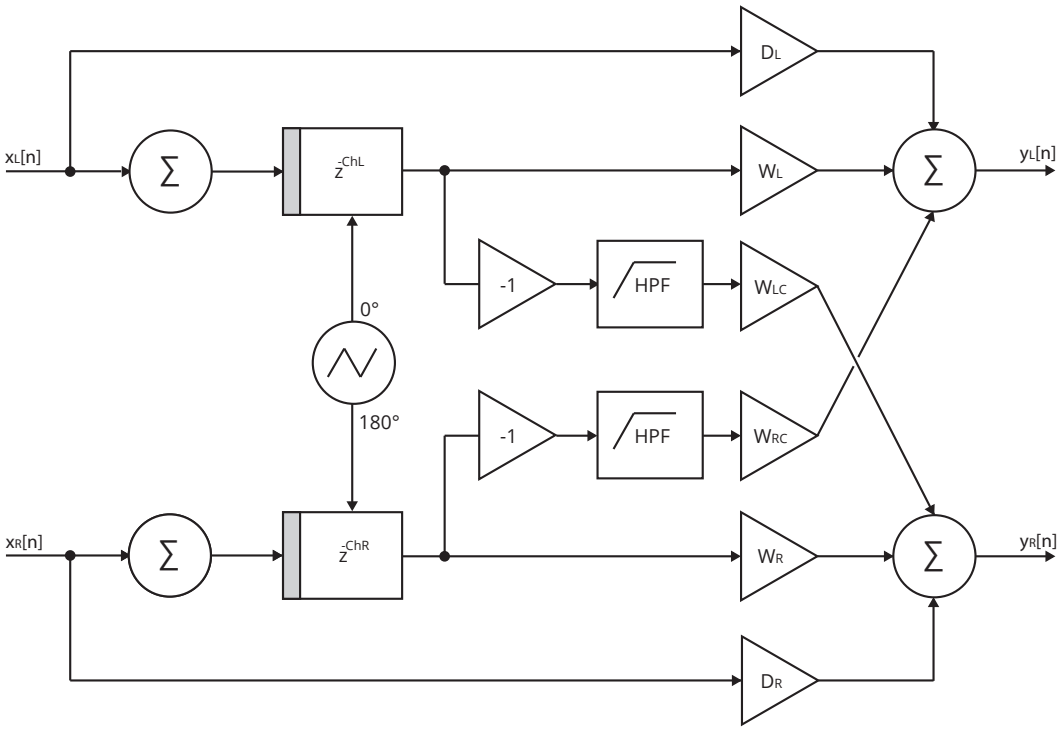
Jak je výše naznačeno, tento princip lze simulovat i s použitím zpožďovací linky s proměnnou délkou zpoždění. Podobně jako vibráto disponuje i flanger parametrem *rate* pro nastavení frekvence oscilátoru a parametrem *depth* pro omezení rozsahu zpoždění, jenž se opět pohybuje v řádu milisekund. Na rozdíl od vibráta se ale ke zpožďovanému signálu přičítá i signál přímý. Síla obou těchto signálů bývá zpravidla nastavena na 50 %, lze se však setkat s flangery, jež dovolují jejich poměr upravit pomocí parametrů *wet* a *dry*. Oproti vibrátu obsahuje flanger i zpětnovazební větev, díky niž lze změnou parametru *feedback* zvukový efekt umocnit [14].

Na obrázku 4.12 můžete vidět klasickou variantu stereo flangeru, která se vyznačuje fázovým posunem řídícího signálu u pravého kanálu o 90°, což vytváří působivý stereofonní efekt. Nejčastěji se setkáváme s použitím flangeru na stopu elektrické kytary, nicméně lze jej kreativně aplikovat například i na bicí či vokály.

4.3.4 Chorus

Poslední efektem založeným na zpožďovací lince s proměnnou délkou zpoždění, který zde zmíním, je *chorus*. Smyslem tohoto efektu je napodobení jevu, který vzniká při sloučení více zvuků podobné barvy a výšky. S tímto jevem se můžeme setkat například u skupiny zpěváků zpívajících unisono. Typický zvuk je způsoben drobnými výchylkami v ladění a časování jednotlivých projevů [30].

Struktura efektu chorus je opět velmi podobná strukturám předcházejících dvou efektů. Délka zpoždění se pohybuje mezi 7 a 40 ms, přičemž podobně jako vibráto nedisponuje zpravidla ani chorus zpětnou vazbou. Na obrázku 4.13 můžete vidět variantu vycházející z analogového efektu Roland Dimension D [14].



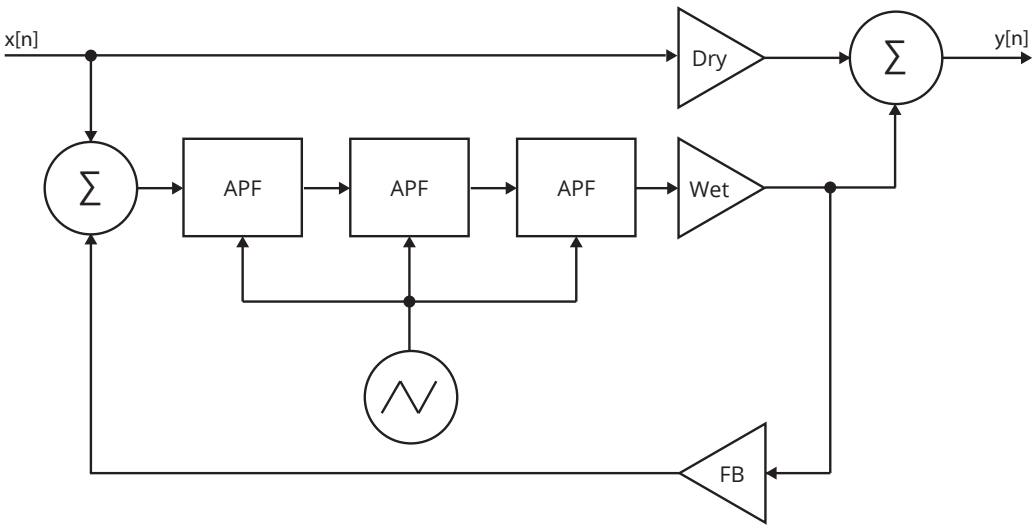
Obrázek 4.13: Chorus vycházející z analogového efektu Roland Dimension D [14].

4.4 Modulované filtry

Podobně jako efekty založené na zpožďovací lince s proměnnou délkou zpoždění využívají i modulované filtry nízkofrekvenční oscilátor pro automatickou změnu jednoho či více parametrů.

4.4.1 Phaser

Phaser je efekt, který kombinuje přímý signál se signálem, jenž prochází sérií fázovacích článků, jejichž střední frekvence je řízena pomocí nízkofrekvenčního oscilátoru [32]. Poměr mezi těmito dvěma signály lze upravit změnou parametrů *wet* a *dry*, přičemž výsledný zvukový efekt je možné zvýraznit prostřednictvím parametru *feedback*. Frekvence řídicího signálu se pohybuje v řádu jednotek Hz.

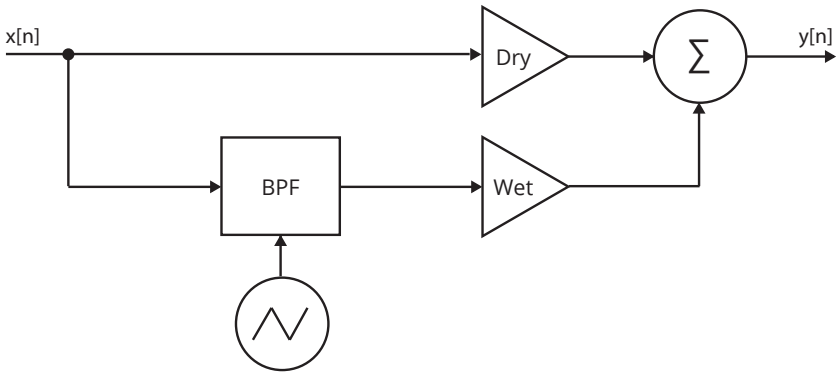


Obrázek 4.14: Blokové schéma efektu phaser [32].

Počet fázovacích článků se liší podle konkrétní implementace phaseru. Lze se dokonce setkat s phasery, jež disponují i více než třiceti fázovacími články. Některé phasery využívají namísto série fázovacích článků sérii filtrů typu notch, čímž dosahují velmi podobných výsledků [17]. Zvuk tohoto efektu se blízce podobá zvuku flangeru, pročež tyto dva efekty bývají často zaměňovány. Nejčastěji se s použitím phaseru můžeme setkat u kláves či elektrických kytar.

4.4.2 Wah-wah

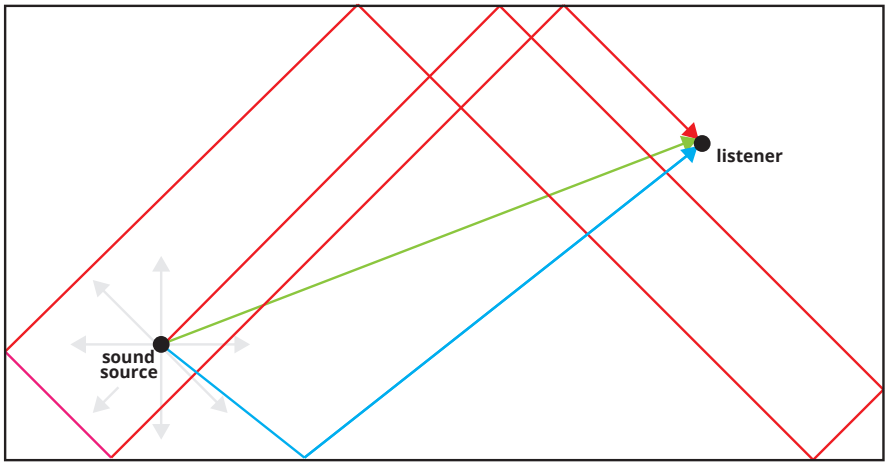
Wah-wah je klasický kytarový efekt, pro který se v českém jazyce vžil název *kvákadlo*. Tento efekt k přímému signálu přičítá signál, jenž prochází pásmovou propustí, přičemž střední frekvenci pásmové propusti může kytarista měnit sešlápnutím nožního pedálu. Zpravidla se tato frekvence pohybuje mezi 400 a 1200 Hz. Na obrázku 4.15 je zobrazena digitální verze tohoto efektu, u které je nožní pedál nahrazen nízkofrekvenčním oscilátorem. Takto upravený efekt bývá často označován jako takzvaný *auto-wah*. Některé implementace využívají místo pásmové propusti filtr typu peak či low-pass [17].



Obrázek 4.15: Blokové schéma efektu auto-wah [17].

4.5 Dozvukové procesory

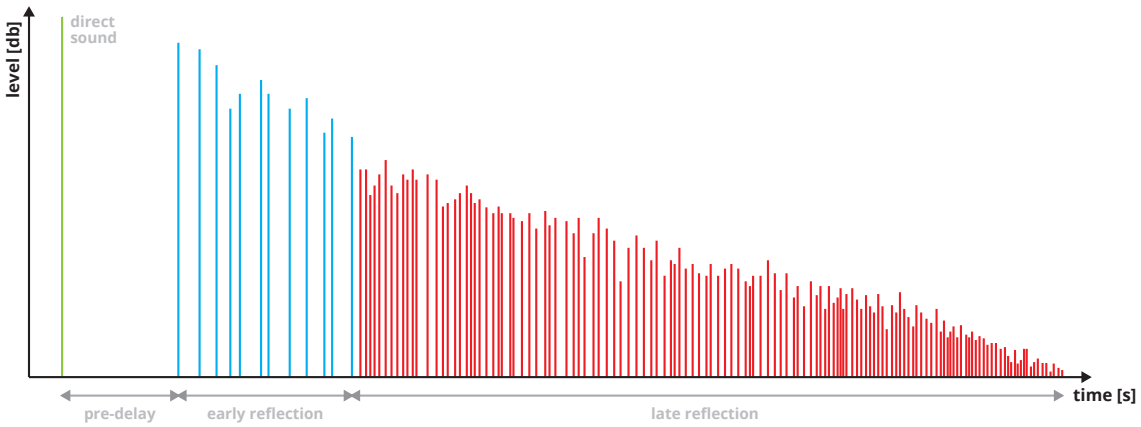
Dozvukové procesory, které jsou také často známy i pod názvem *reverby*, tvoří významnou a oblíbenou kategorii zvukových procesorů, jejichž úkolem je vytvářet umělý dozvuk, napodobující průběh dozvuku ve skutečných uzavřených prostorech, jakými jsou například sály, jeskyně, koncertní haly, katedrály, kluby či velká nahrávací studia. Některé reverby dokonce umožňují napodobit dozvuk i pro člověka mnohem běžnějších prostorů, jako například obývacího pokoje, kanceláře či koupelny [30].



Obrázek 4.16: Ilustrace šíření zvuku od zdroje k posluchačovi v uzavřeném prostoru.

Zvuk se od jeho zdroje šíří všemi směry, pročež posluchač v místnosti či jiném uzavřeném akustickém prostoru slyší kromě přímého zvuku i kombinaci zvuků odražených, jež od zdroje k posluchači urazily delší trajektorii než zvuk přímý a jsou tedy v závislosti na délce této trajektorie opožděny. Zvuk se od stěn, podlahy, stropu či překážek v prostoru odráží pod stejným úhlem, pod jakým na ně dopadá, přičemž část zvuku je povrchem absorbována. Nutno podotknout, že povrch zpravidla neabsorbuje všechny frekvenční složky zvukového signálu stejně. Tento faktor, jenž se se významně podílí i na výsledné barvě zvuku, je ovlivněn především typem materiálu povrchu [17].

Základním parametrem, jenž lze nalézt prakticky u všech dozvukových procesorů je takzvaná *doba dozvuku* (*reverberation time* nebo také *decay time*), často označována jako RT_{60}. Tento parametr udává dobu, za kterou klesne úroveň odrazů o 60 dB, což odpovídá jedné miliontině původní hodnoty výkonu. Doba dozvuku je úzce spjata právě s velikostí uzavřeného prostoru. Zatímco v malých místnostech se doba dozvuku pohybuje v řádu stovek milisekund, v koncertních sálech či katedrálách může dosahovat až několika sekund. Kromě velikosti místnosti ovlivňuje dobu dozvuku opět i typ materiálu povrchu [17].



Obrázek 4.17: Ukázka impulsní charakteristiky přirozeného uzavřeného prostředí.

Na obrázku 4.17 je zobrazena typická impulsní charakteristika přirozeného uzavřeného prostředí, jejíž časový průběh lze rozdělit do tří částí. Časový interval mezi přímým zvukem a jeho prvním odrazem se nazývá *počáteční zpoždění (pre-delay)*. Po počátečním zpoždění následují řídce rozmístěny *počáteční odrazy (early reflection)*, jež nesou informaci zejména o velikosti konkrétního prostoru. Poslední část je pak tvořena stále hustšími odrazy *(late reflection)*, které dotvářejí celkový charakter zvuku [30].

V praxi se můžeme setkat s celou řadou dozvukových procesorů. Rané analogové reverby používané studiovými inženýry v padesátých letech minulého století byly založeny na průchodu zvukového signálu přes kovovou desku či kovové pružiny. S nástupem číslicových technologií se začaly objevovat i první digitální verze reverbů. Nejznámější přístupy a algoritmy digitálních dozvukových procesorů jsou shrnuty v textu následujících podkapitol.

4.5.1 Schroederův reverb

První implementací digitálního dozvukového procesoru zveřejněného již v roce 1961 je takzvaný Schroederův reverb (obr. 4.18), který využívá paralelního zapojení čtyř zpětnovazebních hřebenových filtrů následovaného sériovým zapojením dvou fázovacích článků.

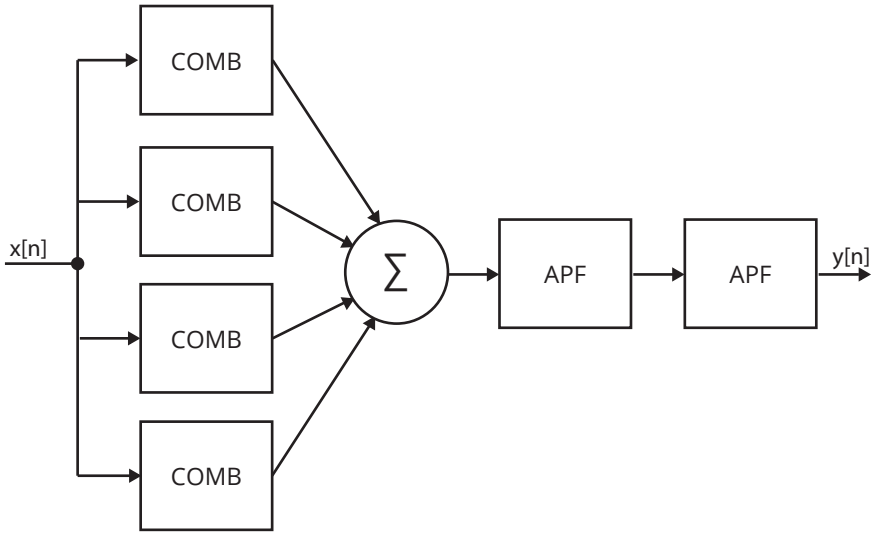
Pro přirozeně znějící barvu umělého dozvuku bez nežádoucího chvění zvuku je jedním ze základních požadavků to, aby hustota odrazů dosahovala hodnoty okolo tisíce odrazů za sekundu. Pokud bychom této hustoty chtěli dosáhnout s použitím výhradně hřebenových filtrů, bylo by jich zapotřebí až několik desítek, což je značně nepraktický přístup, jehož časová složitost by vzhledem k použití systému pro zpracování signálu v reálném čase mohla dosáhnout až hranice použitelnosti [20].

Jak již bylo naznačeno výše, Schroederem navržený model reverbu využívá pro znásobení zhruba sta exponenciálně doznívajících odrazů získaných pomocí čtyř hřebenových filtrů

právě dva fázovací články. Zpoždění hřebenových filtrů se zpravidla pohybuje v rozsahu od 30 do 45 ms, přičemž parametr zeslabení zpětné vazby g lze vypočítat pomocí vztahu:

$$g = 10^{\frac{-3mT_s}{RT_{60}}}, \tag{4.4}$$

kde m je zpoždění hřebenového filtru, T_s vzorkovací perioda a RT_{60} doba dozvuku. Nastavení zpoždění fázovacích článků navrhl Schroeder na 5 a 1.7 ms. Parametr g je pak u obou fázovacích článků nastaven na 0.7 [20]. Základní provedení tohoto algoritmu je monofonní, lze jej však modifikovat i tak, aby poskytoval více než jeden výstup. Tato modifikace zahrnuje zařazení série fázovacích článků před sekci hřebenových filtrů a rozšíření systému o takzvanou *násobící matici*, která definuje jednotlivé výstupní kanály reverbu pomocí různých kombinací přímých a invertovaných výstupů hřebenových filtrů [14].



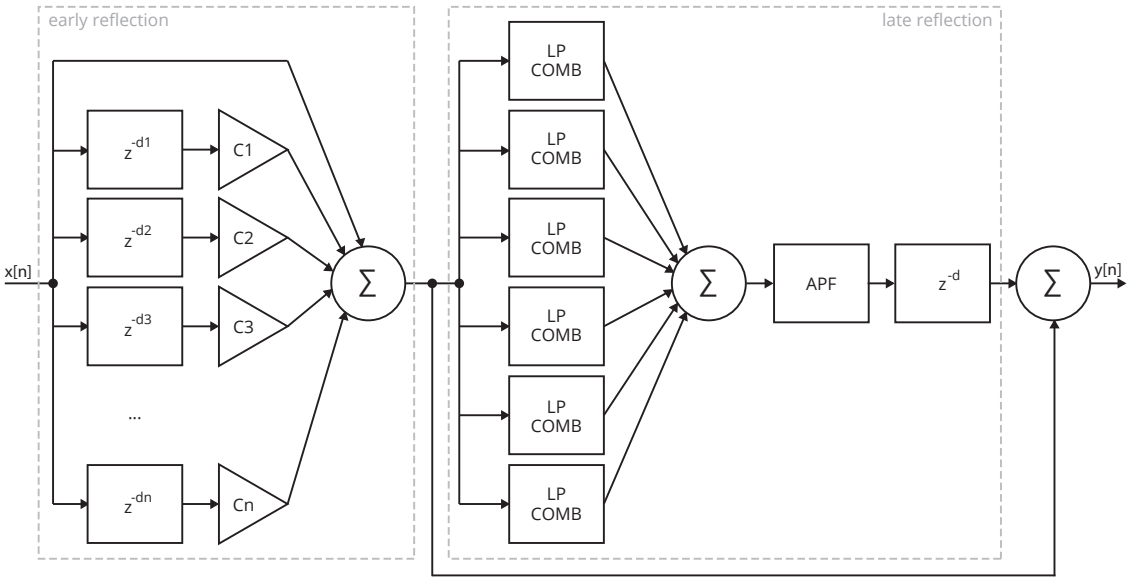
Obrázek 4.18: Blokový diagram Schroederova reverbu.

4.5.2 Moorerův reverb

Dalším důležitým modelem v oblasti dozvukových procesorů je *Moorerův reverb* z roku 1979, jenž svoji strukturou vychází ze Schroederova reverbu. Moorer zkoumal vlastnosti dozvuku Bostonské symfonické síně a četnou řadou pokusů zjistil, že zvukové signály s vyšší frekvencí jsou odrazem od povrchů a třením vzduchu tlumeny mnohem výrazněji než zvuky s frekvencí nižší. Schroederův model tento faktor nebere v úvahu, což v některých případech může vést ke kovově znějícím rezonancím ke konci dozvuku [26].

Na obrázku 4.19 je zobrazen blokový diagram Moorerova reverbu skládajícího se ze dvou částí. První část disponuje sadou paralelně zapojených zpožďovacích linek, jejichž výstupy jsou v různém poměru kombinovány s přímým signálem. Smyslem tohoto zapojení je simulace prvotních odrazů v uzavřeném prostoru. Moorer ve své práci uvádí variantu s šesti zpožďovacími linkami, jež simulují prvotní odrazy v malé místnosti a také variantu s osmnácti zpožďovacími linkami, jejichž zpoždění a následné zeslabení je nastaveno podle výsledků měření z výše zmíněné symfonické síně [12].

Druhá část Moorerova modelu je tvořena kombinací šesti paralelně zapojených zpětnovazebních hřebenových filtrů následovaných frekvenčním článkem (viz sekce 4.1.2) a zpožďovací linkou (viz sekce 4.3). Tato sekce má za úkol simulovat pozdní husté odrazy uzavřeného prostoru. Moorer vyřešil výše uvedený problém s rezonancí vysokých frekvencí postihující Schroederův model tím, že do zpětné vazby každého z hřebenových filtrů přidal dolní propusť. Výsledný zvukový dojem Moorerova dozvukového procesoru pak díky této modifikaci mnohem více připomíná dozvuk reálného uzavřeného prostoru [12].



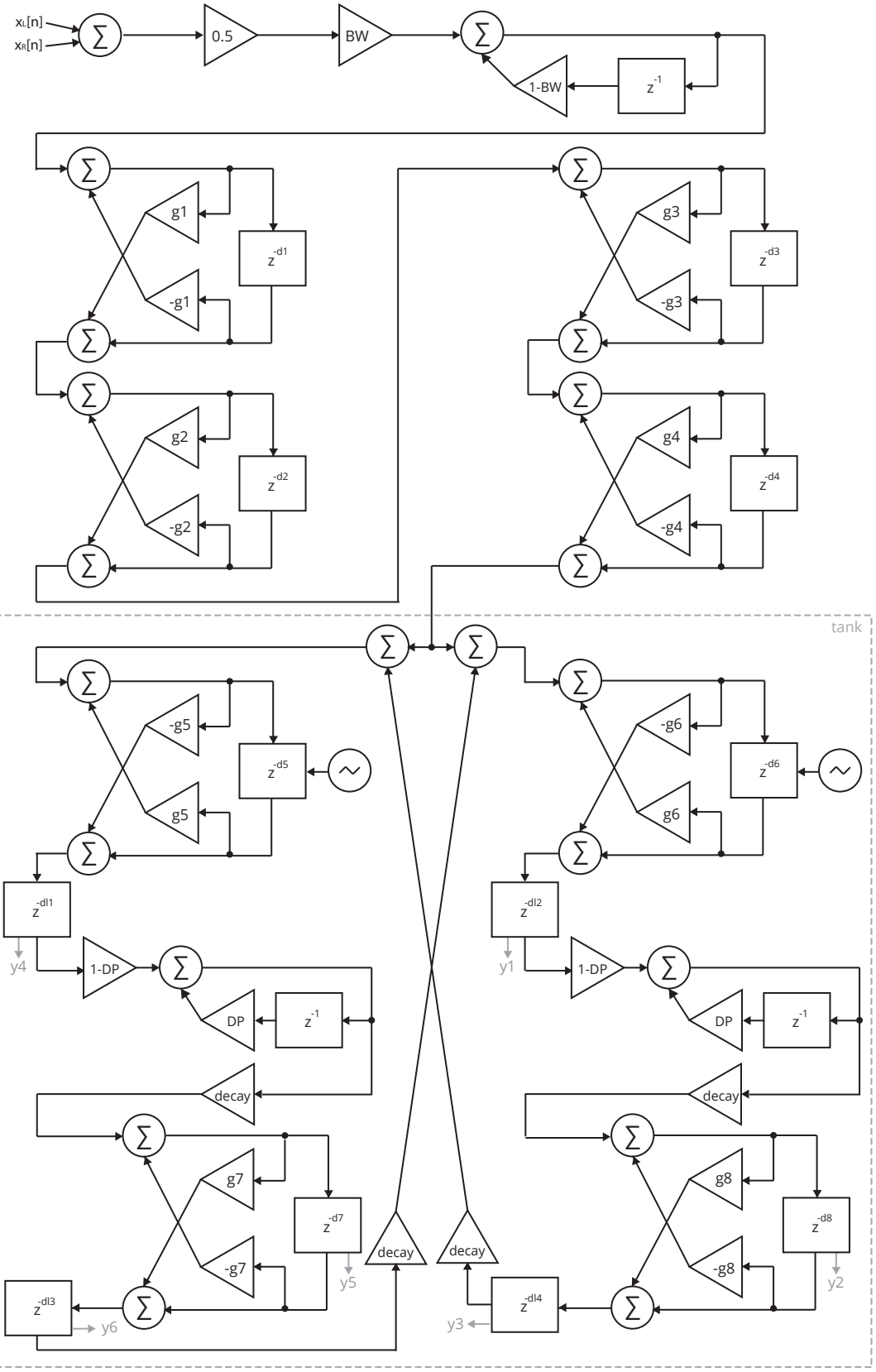
Obrázek 4.19: Blokový diagram Moorerova reverbu [12].

4.5.3 Dattorrův reverb

Posledním dozvukovým procesorem založeným na kombinaci zpožďovacích linek, dolních propustí a fázovacích článků, který v této kapitole zmíním, je takzvaný Dattorrův reverb (obr. 4.20). Oproti předchozím modelům obsahuje tento reverb i další komponentu, kterou je fázovací článek s modulovanou délkou zpoždění. Pro tuto modulaci je podobně jako u dříve zmíněných efektů používán nízkofrekvenční oscilátor.

Dattorrův reverb disponuje stereofonním vstupem, přičemž však zvukový signál z obou vstupních kanálu sčítá a následně dělí dvěma. Posléze signál prochází dolní propustí prvního řádu, jejíž mezní frekvenci lze ovlivnit pomocí parametru *bandwidth*. Dalším krokem je průchod signálu čtveřicí sériově zapojených fázovacích článků, jejichž úkolem je náhodně rozptýlit fázi signálu a tím jej připravit pro další zpracování [2].

Poté je zvukový signál přiveden do struktury zvané *tank*, která příchozí signál rozdělí do dvou paralelních cest. Každá z těchto větví obsahuje do série zapojený fázovací článek s modulovanou délkou zpoždění, zpožďovací linku, dolní propusť prvního řádů ovlivnitelnou pomocí parametru *damping*, fázovací článek a druhou zpožďovací linku. Výstup první větve je přiveden na začátek větve druhé a analogicky výstup druhé větve je přiveden na začátek větve první. Tímto zapojením je vytvořena křížená zpětná vazba, přičemž její velikost lze nastavit pomocí parametru *decay*.



Obrázek 4.20: Blokový diagram Dattorrova reverbu [2].

Pozornému oku čtenáře zajisté nemůže uniknout to, že některé ze zpožďovacích linek disponují i dalším výstupem. Díky těmto pomocným výstupům lze přistupovat na konkrétní pozice kruhových bufferů zpožďovacích linek a tím získat vzorky signálu s požadovaným zpožděním. Tyto vzorky jsou pak využity pro výpočet stereofonního výstupu Dattorrova reverbu prostřednictvím následujících rovnic:

$$\begin{aligned} y_L &= 0.6 \cdot (y_1 + y_1 - y_2 + y_3 - y_4 - y_5 - y_6), \\ y_R &= 0.6 \cdot (y_4 + y_4 - y_5 + y_6 - y_1 - y_2 - y_3). \end{aligned} \tag{4.5}$$

Konkrétní zpoždění pomocných výstupů i nastavení parametrů zpožďovacích linek a fázovacích článků pro různé vzorkovací frekvence jsou podrobně popsány v Dattorrově práci, jenž byla vydána v roce 1997 [2].

4.5.4 Konvoluční reverb

Posledním typem digitálních dozvukových procesorů, který v této kapitole alespoň okrajově zmíním, je takzvaný *konvoluční reverb* [17]. Svůj název si tento typ reverbu získal tím, že je založen na *diskrétní konvoluci*, což je matematická operace nad dvěma signály značena symbolem $\star$ a popsána rovnicí:

$$y[n] = x[n] \star h[n] = \sum_{k=0}^{N-1} h[k]x[n-k], \tag{4.6}$$

kde x je vstupní signál, h impulzní charakteristika uzavřeného prostoru a N její délka.

Existuje mnoho měřících metod pro získání impulsní charakteristiky reálného uzavřeného prostoru, přičemž většina z nichž je založena na vyslání vhodného signálu do prostoru a následném snímání jeho odrazů. Teoreticky ideálním signálem pro tento účel by byl takzvaný *Diracův impuls*, což je nekonečně úzký a nekonečně vysoký signál, jehož integrál je roven jedné. Tento signál však není možné technicky realizovat. V praxi se pro měření impulsní charakteristiky uzavřených prostorů nejčastěji používají krátké obdélníkové impulsy, bílý a růžový šum či signál zvaný *sweep*, což je sinusový signál jehož frekvence je lineárně nebo logaritmicky zvyšována [11].

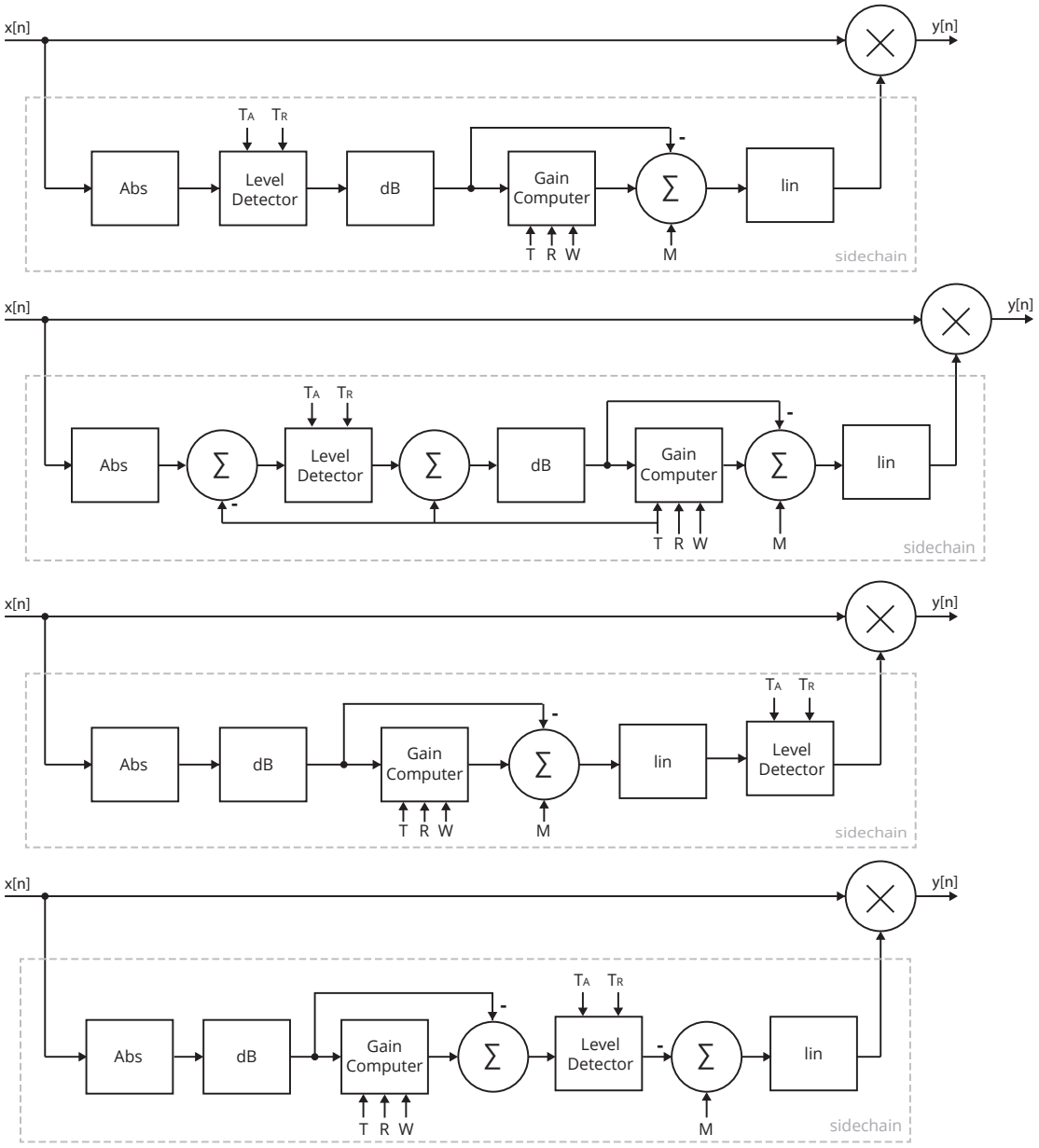
Nespornou výhodou konvolučních dozvukových procesorů je jejich zvuková věrohodnost. Z výše uvedené rovnice však vyplývá, že přímý výpočet diskrétní konvoluce v časové oblasti zahrnuje vysoké množství operací sčítání a násobení, což se negativně projevuje na zvýšené časové složitosti, která lineárně roste s délkou impulzní charakteristiky. Vezmeme-li v úvahu, že impulzní charakteristiky některých reálných uzavřených prostorů mohou dosahovat délky až několika sekund, můžeme se při použití diskrétní konvoluce pro zpracování signálu v reálném čase dostat až za hranici použitelnosti [27].

Tento hendikep lze vyřešit přechodem do frekvenční oblasti, ve které je výsledek konvoluce dvou signálů roven součinu jejich frekvenčních spekter, jež lze získat s využitím algoritmu *rychlé Fourierovy transformace (fast Fourier transform, FFT)* [17].

4.6 Dynamické procesory

Dynamické procesory spadají do kategorie nelineárních zvukových procesorů a jak už jejich název napovídá, slouží k úpravě dynamického rozsahu, jenž je definován jako rozdíl hla-

sitosti nejtišššího a nejhlasitějšího úseku konkrétního zvukového signálu, kterým může být například zpěv či hudební nástroj, ale i kompletní hudební skladba. Jinými slovy tedy dynamické procesory automaticky řídí hlasitost zvukového signálu. Pro zmenšení dynamického rozsahu se používají takzvané *kompresory* a *limitery*, pro zvětšení dynamického rozsahu pak *expandery* a *šumové brány* [13].

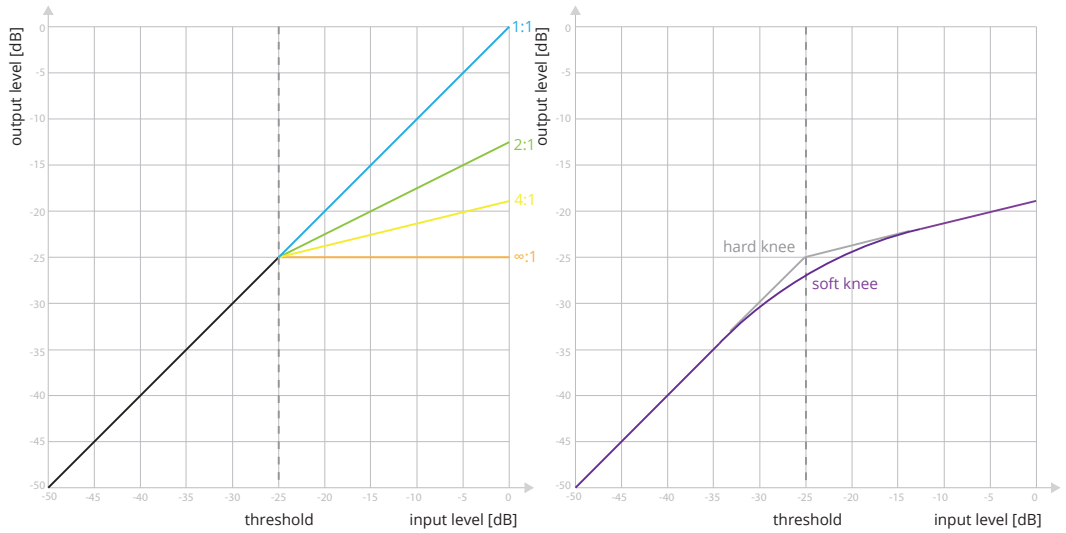


Obrázek 4.21: Struktura dynamického procesoru s různým umístěním detekce úrovně [17].

4.6.1 Kompresory a limitery

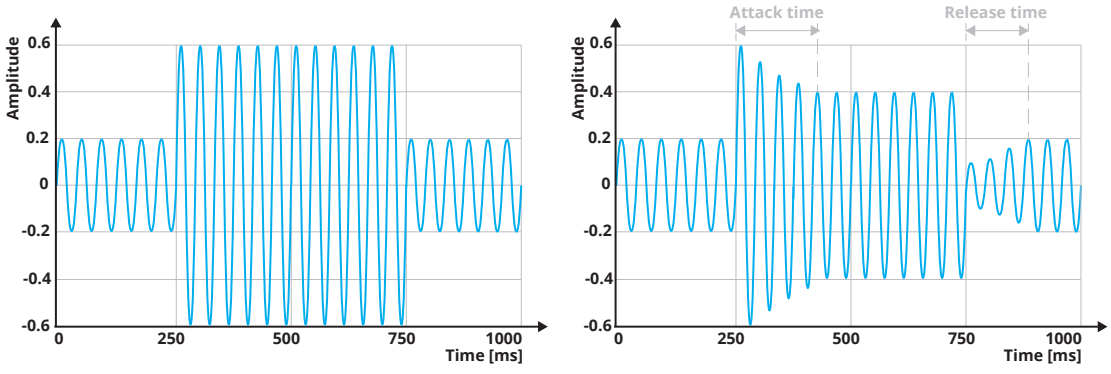
Jak již bylo naznačeno výše, *kompresor* je dynamický procesor, jenž se využívá k omezení dynamického rozsahu zvukového signálu. Základními řídicími parametry kompresoru jsou

práh citlivosti (threshold) a *kompresní poměr (ratio)*. Vstupní zvukový signál, jehož hlasitost je nižší než uživatelem nastavený práh citlivosti, není kompresorem nikterak ovlivněn. Pokud však hlasitost vstupního signálu překročí tuto prahovou úroveň, kompresor sníží přesah dle nastaveného poměru. Pokud je tedy například nastaven kompresní poměr 3:1 a vstupní signál překročí práh citlivosti o 3 dB, kompresor sníží tento přesah na hodnotu 1 dB [30].



Obrázek 4.22: Křivky komprese s různým nastavením parametrů ratio a knee [28].

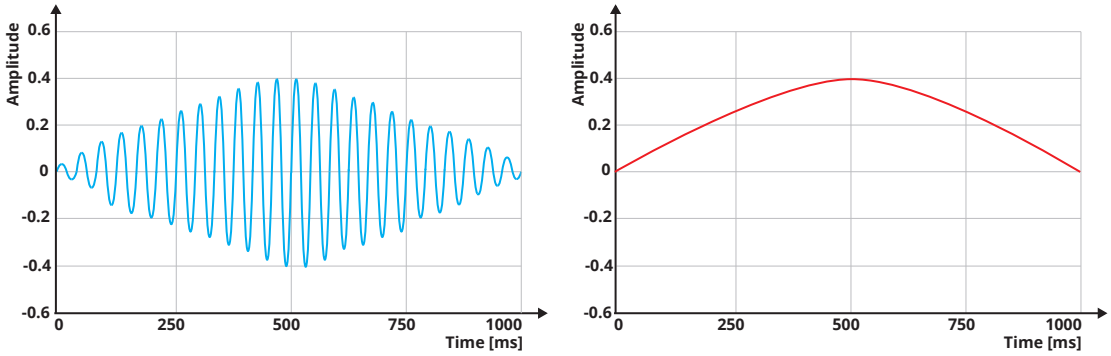
Zalomení kompresní křivky se označuje jako takzvané *koleno (knee)*, přičemž většina moderních digitálních kompresorů umožňuje kromě *ostrého kolene (hard knee)* využiti i *kolene měkkého (soft knee)*, jehož rozsah je zpravidla nastavitelný pomocí parametru *šířka kolene (knee width)*. Většina kompresorů také obvykle disponuje parametry pro nastavení vstupní a výstupní úrovně signálu. Kompresor, jehož kompresní poměr je fixně nastaven na ∞:1, se nazývá *limiter*, nicméně v praxi se pojem limiter vžil pro libovolný kompresor s kompresním poměrem alespoň 10:1. Posledními dvěma běžnými parametry kompresoru jsou *doba náběhu (attack time)* a *doba doběhu (release time)*, viz obrázek 4.23.



Obrázek 4.23: Ukázka nekomprimovaného a komprimovaného signálu.

Struktura digitálního kompresoru sestává ze dvou paralelních signálových cest, viz obrázek 4.21. První signálová větev se nazývá *přímá* a prochází přes násobičku na výstup. Druhá signálová větev je označována jako *řídící (sidechain)* a je tvořena několika dílčími bloky,

jejichž společným úkolem je podle úrovně hlasitosti vstupního signálu vypočítat zeslabující činitel, jenž je přiveden na druhý vstup zmíněné násobičky [19].



Obrázek 4.24: Vstupní signál a odpovídající obálka.

Zvukový signál procházející řídicí větví je nejprve usměrněn pomocí bloku počítajícího absolutní hodnotu a následně vstupuje do bloku pro detekci úrovně signálu, přičemž cílem tohoto bloku je poskytnutí plynulé reprezentace špičkové či efektivní úrovně vstupního signálu. Tato reprezentace bývá velmi často označována jako *obálka (envelope)*. Příklad obálky pro ukázkový vstupní signál je zobrazen na obrázku 4.24. Existuje celá řada jednoduchých i složitějších struktur využitelných pro detekci obálky signálu, viz [9]. Oblíbeným řešením bývá použití vyhlazovacího filtru prvního řádu s nekonečnou impulzní odezvou, který je popsán diferenční rovnicí:

$$y[n] = \begin{cases} \alpha_A y[n-1] + (1-\alpha_A)x[n] & x[n] > y[n-1] \\ \alpha_R y[n-1] + (1-\alpha_R)x[n] & x[n] \leq y[n-1] \end{cases}, \tag{4.7}$$

kde x je vstupní signál, y výstupní signál a α_A respektive α_R jsou koeficienty filtru získané z doby náběhu τ_A respektive doby doběhu τ_R a vzorkovací frekvence f_s pomocí rovnice:

$$\alpha = e^{-1/(\tau f_s)}. \tag{4.8}$$

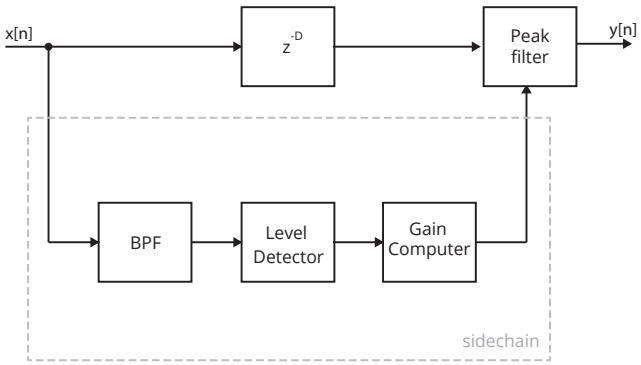
Výstup detektoru obálky je následujícím blokem konvertován do logaritmické podoby a poté přiveden na vstup bloku, jehož úlohou je zeslabení obálky signálu podle vztahu:

$$y = \begin{cases} x & 2(x-t) < -w \\ x + (1/r - 1)(x - t + w/2)^2/(2w) & 2|(x-t)| \leq w \\ t + (x-t)/r & 2(x-t) > w \end{cases}, \tag{4.9}$$

kde x je výstup detektoru obálky, y zeslabená obálka, t práh citlivosti, w šířka kolene a r kompresní poměr. Výsledné zeslabení vstupního signálu je na závěr vypočítáno odečtením původní obálky od zeslabené obálky, přičtením výstupního zesílení a následným převedením výsledku zpět z logaritmické do lineární podoby [17].

Na obrázku 4.21 jsou kromě výše popsané struktury zobrazeny také další tři, které se nejen od té první, ale i mezi sebou navzájem, liší umístěním detektoru obálky v rámci řídicí větve. Za zmínku stojí především poslední varianta, u které je vyhlazování prováděno až po výpočtu zeslabení vstupního signálu a to ještě před zpětným převodem z logaritmické do lineární podoby. Tím je dosaženo pro lidské ucho přirozeněji znějícímu průběhu zeslabení. Všechny uvedené varianty spadají do kategorie takzvaných *dopředných (feedforward)* kompresorů, kromě nichž se můžeme setkat také s kompresory *zpětnovazebními (feedback)*. Pro podrobnější popis obou typů kompresorů a jejich variant si čtenáře dovolím odkázat na zdroj [17], ze kterého jsem při psaní této podkapitoly nejvíce čerpal.

Analogové i digitální kompresory nacházejí dlouhodobě široké uplatnění při postprodukci populární hudby a to jak při mixáži jednotlivých zvukových stop, tak i při masteringu kompletní hudební skladby. Použitím kompresoru či limiteru se při masteringu dosahuje zvýšení průměrné hlasitosti hudební skladby, avšak za cenu snížení jejího dynamického rozsahu, s čímž je úzce spjata takzvaná *válka hlasitostí (loudness war)* [17]. Drtivá většina posluchačů totiž při poslechu dvou totožných signálu, ze kterých je jeden zesílen, označí jako lépe znějící právě ten hlasitější. Tento fenomén je pak hojně využíván hudebními vydavatelstvími v rámci konkurenčního boje o dosažení co nejvyšší průměrné hlasitosti vydávaných skladeb. Kromě snížení dynamického rozsahu však může přílišná komprese způsobit i ztrátu barvy, případně zapříčinit výskyt nežádoucích zvukových artefaktů.



Obrázek 4.25: Struktura dynamického procesoru de-esser [19].

Na závěr této podkapitoly alespoň okrajově uvedu několik dynamických procesorů, jež vznikly odvozením od základní struktury kompresoru. Pokud je do přímé větve zařazena jednoduchá zpožďovací linka, je tento typ kompresoru označován jako *kompresor s nulovou reakcí (lookahead)* [30], neboť dokáže docílit nulového či dokonce záporného času reakce. Zmínku si zajisté zaslouží i *vícepásmový (multiband) kompresor* [22], který, jak už jeho název napovídá, umožňuje uživateli upravovat dynamiku zvukového signálu v několika frekvenčních pásmech.

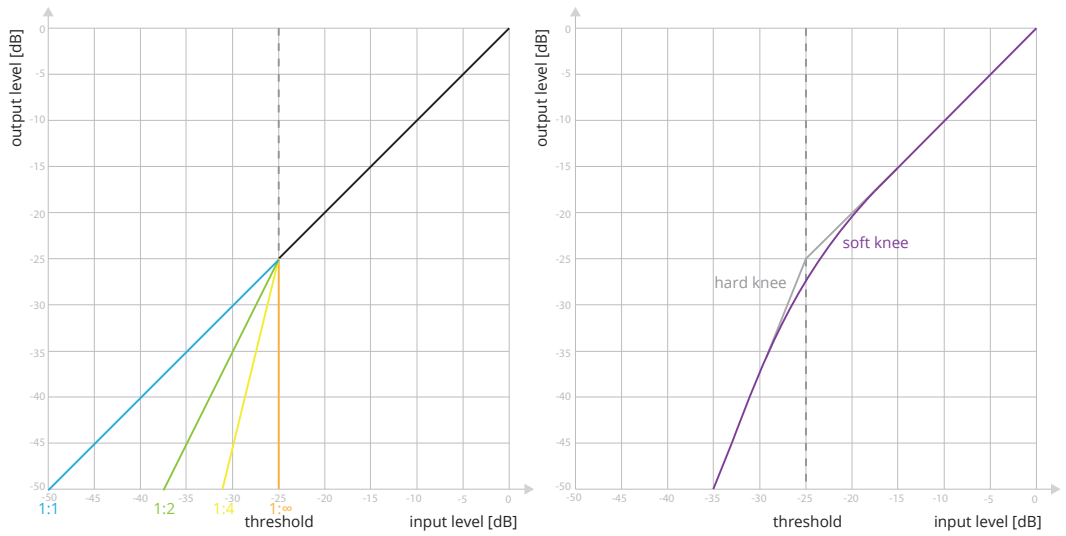
Dalším užitečným nástrojem připomínajícím vícepásmový kompresor je *dynamický ekvalizér* [22], což je parametrický ekvalizér disponující filtry, jež pracují v závislosti na hlasitosti vstupního signálu. Posledním procesorem, jehož blokový diagram je zobrazen na obrázku 4.25, je *de-esser* [19]. De-esser je dynamický procesor, který je určený k potlačení sykavek u vokálních stop, jakými jsou například zpěv, rap či mluvené slovo. Namísto násobičky je ve struktuře de-esseru použit filtr typu peak, jehož útlum je řízen řídící větví, která je zkraje obohacena o pásmovou propust.

4.6.2 Expandery a šumové brány

Expander je dynamický procesor, který v daném poměru zeslabuje vstupní signál, jehož úroveň hlasitosti je nižší než stanovený práh citlivosti. Extrémním případem expanderu je takzvaná *šumová brána (gate)*, jejíž poměr je fixně nastaven na 1:∞ [17]. Na rozdíl od kompresorů a limiterů, které jsou používány pro snížení dynamického rozsahu signálu, slouží tedy expandery a šumové brány k jeho rozšíření. Struktura obou těchto typů procesorů vychází opět ze struktur uvedených na obrázku 4.21, přičemž funkce zeslabení je dána vztahem:

$$y = \begin{cases} t + (x-t)r & 2(x-t) > w \\ x - (r-1)(x-t-w/2)^2/(2w) & 2|(x-t)| \leq w \\ x & 2(x-t) < -w \end{cases}, \tag{4.10}$$

kde x je výstup detektoru obálky, y zeslabená obálka, t práh citlivosti, w šířka kolene a r poměr expanze. Expandery a šumové brány jsou hojně využívány například při mixáží bicích. Bicí souprava skládající se z několika bubnů je zpravidla snímána více mikrofony. Stopa velkého či malého bubnu však často obsahuje slabé přeslechy činelů, jejichž ruční potlačení by bylo příliš pracné. Dalším častým využitím expanderu či šumové brány je redukce šumu, případně nádechů, u vokálních stop, zejména pak u mluveného slova.



Obrázek 4.26: Křivky expanze s různým nastavením parametrů ratio a knee [28].

Kapitola 5

Grafy

Graf je základní abstraktní matematická struktura *teorie grafů*, která často nachází uplatnění nejen v matematice, ale také informatice, geografii a mnoha dalších vědních disciplínách. Skládá se z množiny prvků, které nazýváme *uzly* či *vrcholy* a množiny *hran*, jež reprezentují vztahy mezi uzly. Graf je možné formálně popsat pomocí [29]:

Definice 5.1. Graf je trojice $G = (H, U, \rho)$, kde

H je množina hran $H = \{h_1, h_2, \ldots, h_n\}$

U je množina uzlů $U = \{u_1, u_2, \ldots, u_n\}$

ρ je incidenční zobrazení, které je dáno předpisem

pro neorientovaný graf
$H \to U \otimes U$, kde $U \otimes U$ označuje množinu všech neuspořádaných dvojic prvků z množiny U

pro orientovaný graf
$H \to U \times U$, kde $U \times U$ označuje kartézský součin (množinu všech uspořádaných dvojic prvků z množiny U).

V literatuře se často setkáváme i s touto alternativní zjednodušenou definicí grafu [5]:

Definice 5.2. Graf je dvojice $G = (U, H)$, kde

U je množina uzlů $U = \{u_1, u_2, \ldots, u_n\}$

H je množina hran

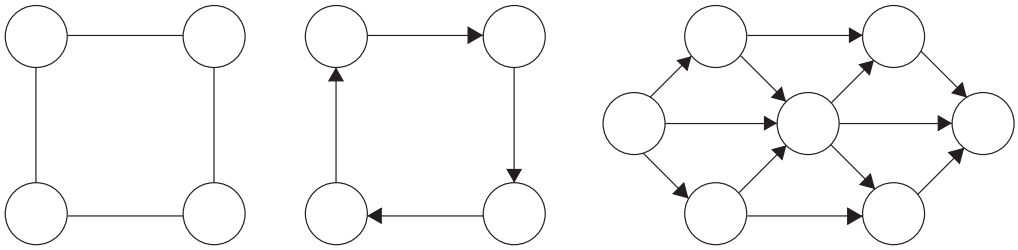
pro neorientovaný graf
$H \subseteq \{\{u, v\} \mid u, v \in U\}$

pro orientovaný graf
$H \subseteq U \times U$

5.1 Detekce cyklu v orientovaném grafu

Jak již výše uvedené definice naznačují, grafy lze rozdělit na *orientované* a *neorientované*, podle toho zda-li jsou orientované jejich hrany či ne. Orientace hran bývá zpravidla graficky znázorněna pomocí šipek. *Kružnice* je souvislý neorientovaný graf, pro jehož každý

uzel platí, že má právě jednu vstupní a jednu výstupní hranu. Obdobou *kružnice* u orientovaných grafů je *cyklus*, viz obrázek 5.1. Pokud orientovaný graf obsahuje cyklus jako svůj podgraf, označujeme jej jako *cyklický*, v opačném případě jako *acyklický*. Pro detekci cyklu v orientovaných grafech slouží algoritmy, jež jsou uvedeny v následujících sekcích [6].



Obrázek 5.1: Zleva: kružnice, cyklus, acyklický graf.

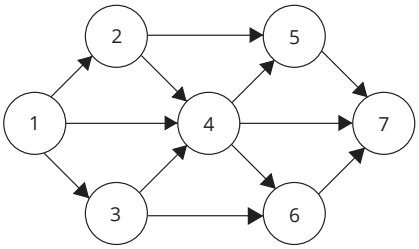
5.1.1 Topologické číslování uzlů

Věta 5.1. Graf $G = (U, H)$ je acyklický právě tehdy, lze-li všechny uzly z množiny U očíslovat čísly $1, 2, \ldots, n$ tak, že pro každou hranu $(i, j) \in H$ platí, že $i < j$ [21].

Důkaz předcházející věty vychází z faktu, že každý podgraf acyklického grafu je také acyklický. Uzel, jenž není koncový pro žádnou hranu grafu, se nazývá *pramen*, přičemž je zřejmé, že acyklický graf obsahuje minimálně jeden pramen. Z opačného případu by totiž plynula existence cyklu v grafu.

Algoritmus 5.1. Topologické číslování uzlů [5]

1. Položme $G_1 = G$ a $k = 1$.
2. V grafu G_k nalezneme uzel, který nemá žádné vstupní hrany a přidělíme mu číslo k. Pokud takový uzel neexistuje, znamená to, že graf G není acyklický a postup končí.
3. Z grafu G_k vytvoříme graf G_{k+1} tak, že z něj vypustíme uzel s číslem k a všechny hrany, které z tohoto uzlu vychází.
4. Je-li graf G_{k+1} neprázdný, položíme $k = k + 1$ a vrátíme se k bodu 2. V opačném případě postup končí, protože všechny uzly se podařilo topologicky očíslovat a graf G je tedy acyklický.



Obrázek 5.2: Acyklický graf z obrázku 5.1 s topologicky očíslovanými uzly.

5.1.2 Detekce cyklu s využitím DFS

Prohledávání do hloubky (depth-first search, DFS) je jeden ze základních grafových algoritmů, který lze využít i jako součást postupu pro detekci cyklu v orientovaném grafu. Následující algoritmus pracuje se třemi navzájem disjunktními množinami uzlů. Množina *white* obsahuje doposud neprozkoumané uzly, množina *grey* uzly právě prozkoumávané a množina *black* uzly, které již byly včetně jejich potomků prozkoumány. Graf je acyklický v případě, že se podaří přesunout všechny uzly z množiny *white* do množiny *black* [1].

Algoritmus 5.2. Detekce cyklu s využitím DFS [7]

bool main:

1. Vytvoř prázdné množiny *white*, *grey* a *black*.
2. Do množiny *white* vlož všechny uzly grafu $G = (U, H)$.
3. Vyber libovolný uzel $w \in white$ a zavolej metodu $r = DFS(w)$.
   3.1. Pokud $r == true$ vrať $true$.
   3.2. Jinak pokud $white == \emptyset$, přejdi na krok 4.
   3.3. Jinak pokračuj dalším uzlem w.
4. Všechny uzly náleží množině *black*, graf neobsahuje cyklus, vrať $false$.

bool DFS(u):

1. Přesuň uzel u z množiny *white* do množiny *grey*.
2. Pro každý uzel v, takový že $(u, v) \in H$.
   2.1. Pokud uzel $v \in grey$, pokračuj dalším uzlem v.
   2.2. Jinak pokud uzel $v \in black$, graf obsahuje cyklus, vrať $true$.
   2.3. Jinak rekurzivně zavolej metodu $r = DFS(v)$.
      2.3.1. Pokud $r == true$ vrať $true$.
      2.3.2. Jinak pokračuj dalším uzlem v.
3. Přesun uzel u z množiny *grey* do množiny *black*.
4. Vrať $false$.

Kapitola 6

Implementace

V této kapitole se zabývám implementací audio pluginu *Ptolemaios* v programovacím jazyce C++ s využitím aplikačního frameworku *JUCE*. Plugin zpracovává digitální zvukový signál v reálném čase a jak už bylo řečeno dříve, je primárně určený pro postprodukci populární hudby. V rámci tohoto pluginu je uživateli poskytnuta kolekce zvukových efektů a procesorů, z nichž si uživatel jejich propojením může podle svých potřeb vytvořit vlastní originální multiefekt. Algoritmy zvukových efektů a procesorů byly podrobně popsány v kapitole 4. Kromě literatury, jejíž seznam je uveden v závěru práce, jsem během implementace pluginu čerpal také z následujících elektronických zdrojů:

www.g200kg.com jsou webové stránky obsahující galerii volně stažitelných vizualizačních a ovládacích prvků vytvořených pomocí aplikace *KnobMan*. Tato aplikace je snadno použitelný freewarový multiplatformní grafický editor, jenž je také k dispozici volně ke stažení.

www.musicdsp.org je jedním z nejstarších webů zabývajících se zpracováním digitálního zvukového signálu. Na tomto webu je umístěna rozsáhlá sbírka úryvků zdrojových kódů nashromážděných komunitou, které lze volně využít k řešení některých praktických problémů zpracování digitálního zvuku. Tyto webové stránky jsou bohužel však již několik let neaktualizovány.

www.willpirkle.com jsou webové stránky věnované tvorbě digitálních zvukových efektů a syntezátorů s využitím aplikačního frameworku *RackAFX*. Autorem těchto webových stránek i zmíněného frameworku je Will Pirkle, který je zároveň i docentem na University of Miami, kde vyučuje audio inženýrství. Will Pirkle je také autorem dvou obsáhlých knih [14, 15] zabývajících se vývojem zvukových efektů a syntezátorů na platformě RackAFX.

www.kvraudio.com je v současnosti největším webovým portálem zabývajícím se situací na poli digitálních zvukových efektů. Tento portál disponuje rozsáhlou databází obsahující podrobný popis několika tisíců pluginů včetně informací o jejich vývojářích, podpoře jednotlivých formátů a v případě komerčních pluginů i o jejich ceně. Součástí portálu je i poměrně živé diskuzní fórum, přičemž jedna z jeho větví se zaobírá právě vývojem audio pluginů.

www.mda-vst.com jsou webové stránky věnované sérii jednoduchých audio pluginů *mda (maximum digital audio)*, které jejich autor Paul Kellett poskytl jako opensource ve formátech VST a AU ke stažení.

6.1 Softwarové technologie

Tato sekce seznamuje čtenáře s technologiemi umožňujícími práci s digitálním zvukovým signálem v reálném čase. Kromě technologií *ASIO* a *MIDI* jsou zde krátce popsány i všechny rozšířené formáty audio pluginů a aplikační framework *JUCE*, jenž byl využit pro implementaci pluginu *Ptolemaios*.

Audio Stream Input/Output

Audio Stream Input/Output (ASIO) [24] je multiplatformní a multikanálový protokol, specifikovaný německou firmou *Steinberg*[1], určený pro ovladače zvukových karet, které oproti běžným ovladačům umožňují hostitelské aplikaci přímé využití hardwaru zvukové karty s minimální asistencí operačního systému, čímž výrazně snižují zpoždění zpracovávaného zvuku. Právě díky nízké latenci mohou profesionální muzikanti a studioví inženýři spolehlivě pracovat se zvukem v reálném čase. Konkrétní ovladače ASIO jsou zpravidla dodávány výrobcem společně se zvukovou kartou, nicméně lze použít i alternativu v podobě volně stažitelných univerzálních ASIO ovladačů.

Obrázek 6.1: Povinná loga standardů ASIO a VST od firmy Steinberg.

Musical Instrument Digital Interface

Musical Instrument Digital Interface (MIDI) [8] je mezinárodní technický standard z roku 1981, jenž specifikuje protokol, digitální rozhraní a podobu konektorů. MIDI umožňuje elektronickým hudebním zařízením komunikovat v reálném čase s počítačem nebo navzájem mezi sebou prostřednictvím sériového rozhraní zasíláním MIDI zpráv. Příkladem praktického využití této technologie je ovládání *digitální pracovní stanice (Digital Audio Workstation, DAW)* pomocí kontroléru či hra na klávesy. Nutno podotknout, že MIDI zprávy neobsahují žádný zvuk, ale pouze řídící informace (síla úderu, výška a délka tónu), na jejichž základě je příslušný zvuk například pomocí syntetizéru či zvukové karty generován.

Virtual Studio Technology

Virtual Studio Technology (VST) je dalším standardem firmy *Steinberg*. Tato technologie byla publikována v roce 1996 a popisuje rozhraní mezi hostitelskou aplikací a zásuvnými moduly, takzvanými *pluginy*. Zásuvné moduly lze rozdělit na dvě kategorie. První kategorii tvoří *VST efekty*, které na vstup dostávají audio signál, jenž modifikují a následně jej předávají na výstup. Do druhé kategorie řadíme *VST instrumenty (VSTi)*, které na vstupu na rozdíl od pluginů z předchozí kategorie očekávají MIDI zprávy, na jejichž základě generují výstupní signál. Na některé pluginy, například vokodér či auto-tune, lze nahlížet jako na zástupce obou zmíněných kategorií.

[1] https://www.steinberg.net

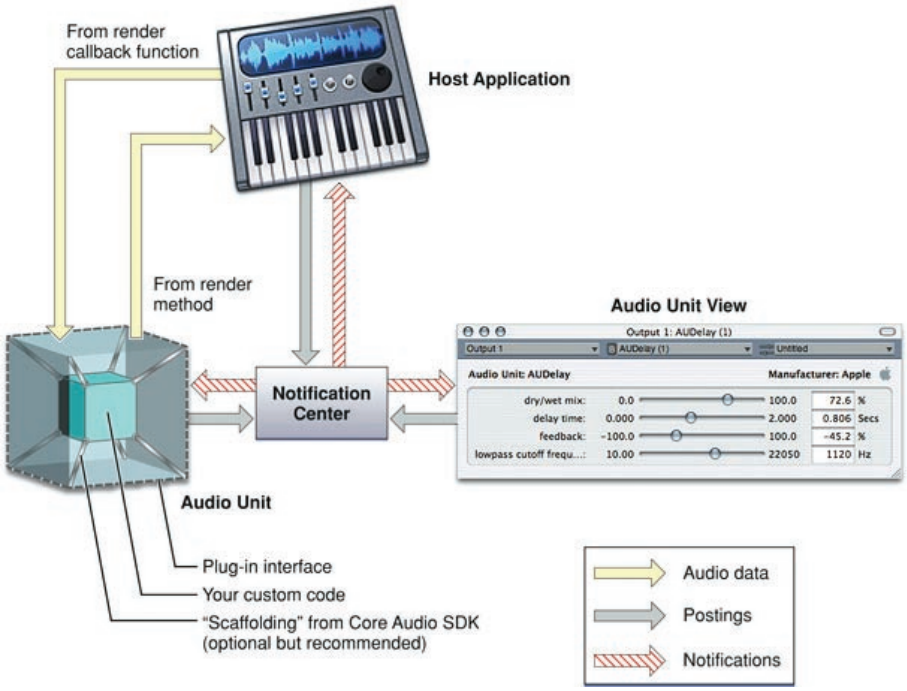
Architektura VST modulů se skládá ze dvou komponent. Komponenta *processor* dle typu modulu zajišťuje provádění výpočetních algoritmů či generování signálu, komponenta *editor* pak poskytuje mechanismy pro nastavování parametrů a komunikaci s grafickým uživatelským rozhraním. Z pohledu hostitelské aplikace se tedy plugin tváří jako černá skříňka definována počtem vstupů a výstupů a nastavitelnými parametry. VST standard je multiplatformní, nicméně nachází uplatnění především na zařízeních s operačním systémem *Windows*. Aktuální verzí je *VST 3.6*, přičemž její SDK lze volně stáhnout z oficiálních stránek firmy *Steinberg* [25].

Avid Audio eXtension

Avid Audio eXtension (AAX) je formát audio pluginů vyvíjený americkou firmou *Avid Technology*[2] (dříve *Digidesing*), který je primárně určený pro nasazení v hostitelských aplikacích řady *Pro Tools*, běžících na operačních systémech *Windows* a *macOS*. Avid Audio eXtension v roce 2013 nahradil dříve používaný formát *Real-Time AudioSuite (RTAS)*. *AAX SDK* je možné po vyplnění krátkého formuláře volně stáhnout včetně podrobné dokumentace z oficiálního webu společnosti *Avid Technology*.

Audio Unit

Posledním běžně používaným formátem pluginů je formát *Audio Unit (AU)*, jenž je založený na technologii *Core Audio*. AU je na rozdíl od výše uvedených formátů určen výhradně pro zařízení s operačními systémy *macOS* a *iOS* od firmy *Apple*. Architektura AU pluginu je znázorněna na obrázku 6.2.



Obrázek 6.2: Architektura Audio Unit, převzato z [10].

[2] http://www.avid.com/

Framework JUCE

JUCE je multiplatformní open-source framework pro vývoj desktopových a mobilních aplikací v programovacím jazyce C++. Podobně jako frameworky *Qt* či *GTK+* disponuje i JUCE bohatou sbírkou tříd poskytujících programátorovi potřebné funkce pro práci s prvky uživatelského rozhraní, grafikou, sítěmi a podobně. Oproti obdobným frameworkům však JUCE vyniká rozsáhlou funkcionalitou z oblasti audia, zejména pak třídami umožňujícími překlad do všech výše zmíněných formátů audio pluginů [18].

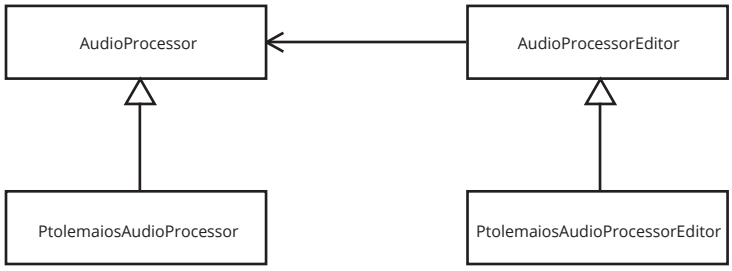
Zdrojové kódy včetně podrobné dokumentace jsou k dispozici na oficiálních stránkách projektu JUCE[3]. Součástí balíčku se zdrojovými kódy je i sada ukázkových příkladů využití frameworku, jež demonstrují řešení konkrétních problémů. Kromě nich balíček obsahuje také užitečný nástroj *Projucer*, který výrazným způsobem usnadňuje správu projektů všech podporovaných platforem. Výhodou je i aktivní komunita uživatelů, kteří komunikují mezi sebou a s vývojáři prostřednictvím oficiálního fóra frameworku JUCE a přispívají tak k jeho zdokonalování.

Obrázek 6.3: Povinné logo frameworku JUCE.

6.2 Základní struktura pluginu

Audio plugin *Ptolemaios* byl vytvořen v programovacím jazyce C++ s využitím aplikačního frameworku JUCE a skládá se ze dvou základních spolupracujících komponent. První z nich je takzvaný *procesor* představovaný třídou `PtolemaiosAudioProcessor`, která vznikla odvozením od třídy `AudioProcessor`. Hlavním úkolem procesoru je zpracování vstupního zvukového signálu.

Druhou komponentou tvořící grafické uživatelské rozhraní je takzvaný *editor*, který je reprezentovaný třídou `PtolemaiosAudioProcessorEditor`, jež vznikla odvozením od třídy `AudioProcessorEditor`. Pro každý spuštěný plugin vytváří hostitelská aplikace vždy pouze jeden procesor, přičemž editor je zpravidla vytvářen až v případě potřeby, tedy může i nemusí v daném okamžiku existovat. Každý editor disponuje referencí na odpovídající procesor, čímž mu je umožněno přistupovat k jeho datům a parametrům.



Obrázek 6.4: Základní struktura pluginu Ptolemaios.

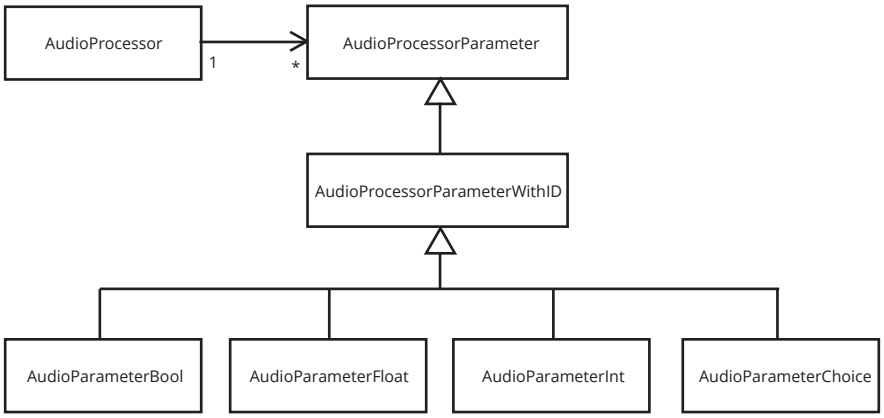
[3] https://www.juce.com

6.3 Procesor pluginu

Jak už bylo naznačeno výše, *procesor* obsahuje algoritmy pro zpracování zvukového signálu. Před začátkem zpracování je hostitelskou aplikací zavolána metoda `prepareToPlay`, jejímiž parametry jsou vzorkovací frekvence a maximální velikost bloku vzorků. Samotné zpracování signálu pak probíhá opakovaným voláním metody `processBlock`, které je jako jeden z parametrů předána reference na objekt třídy `AudioBuffer`; ten obsahuje zvuková data včetně informací o počtu kanálů a aktuální velikosti bloku vzorků. Nutno podotknout, že stejná velikost bloku vzorků není při každém volání této metody garantována. Někdy dokonce může hostitelská aplikace předat procesoru blok, jenž neobsahuje žádné vzorky. Po skončení zpracování signálu volá hostitelská aplikace metodu `releaseResource`, v níž má procesor možnost uvolnit veškeré své alokované zdroje.

Zvukové algoritmy jsou ovlivnitelné nastavením jednotlivých parametrů, jejichž kolekcí procesor disponuje. Základní komponentou reprezentující parametr procesoru je abstraktní třída `AudioProcessorParameter`, z níž jsou pro různé typy parametrů odvozeny třídy uvedené v následujícím výčtu:

`AudioParameterBool`	parametr nabývající hodnoty `true` či `false`
`AudioParameterFloat`	parametr nabývající hodnoty z daného intervalu
`AudioParameterInt`	parametr nabývající celočíselné hodnoty z daného intervalu
`AudioParameterChoice`	parametr nabývající pojmenované hodnoty z daného výčtu



Obrázek 6.5: Třídy pro různé typy parametrů procesoru.

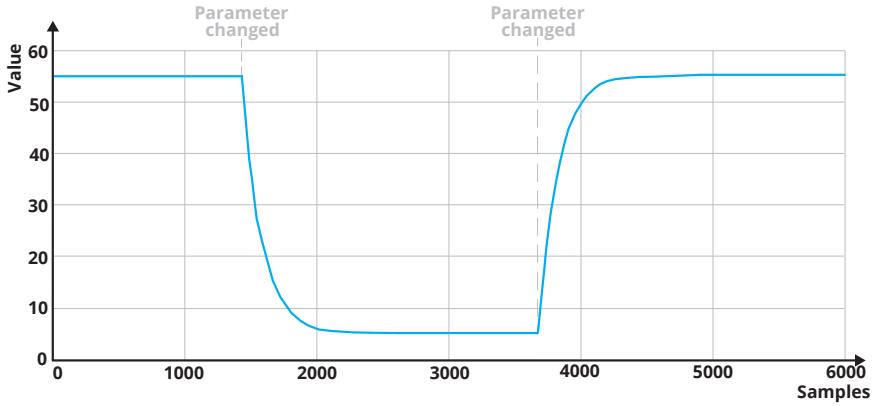
Prvním problémem, se kterým jsem se při implementaci pluginu setkal, byla skutečnost, že většina hostitelských aplikací nepodporuje dynamické přidávání a odebírání parametrů za běhu. Jinými slovy, počet parametrů musí být zpravidla určen už při vytváření instance procesoru a nesmí se již během jeho životního cyklu měnit. Jelikož se v mém případě jedná o multiefekt skládající se z více modulů, jež lze dynamicky přidávat a odebírat, bylo zapotřebí nejdříve rozhodnout, jak k tomuto problému přistoupit.

Jedním z možných řešení by bylo nezahrnovat parametry jednotlivých modulů do kolekce parametrů procesoru. Toto řešení je sice jednoduché, ale připraví uživatele o možnost automatického řízení vybraných parametrů pomocí hostitelské aplikace. Automatické řízení parametru, neboli takzvaná *automatizace*, je však v profesionální postprodukci hudby často využívaná technika, pročež je toto řešení prakticky nepoužitelné.

Jelikož se hostitelská aplikace dotazuje procesoru pluginu na počet parametrů bezprostředně po vytvoření jeho instance, je nutné všechny automatizovatelné parametry definovat již v rámci konstruktoru procesoru. Z tohoto důvodu jsem se rozhodl vyřešit zmíněný problém určením maximálního počtu vložených modulů každého typu. Takto je možné potřebné parametry v konstruktoru procesoru definovat a parametry vkládaných modulů na ně dynamicky za běhu mapovat.

6.3.1 Třída Module

Třída `Module` je odvozena od třídy `AudioProcessor` a slouží jako abstraktní třída, z níž jsou odvozeny konkrétní třídy reprezentující zvukové efekty a procesory, s jejichž algoritmy jsem čtenáře seznámil v kapitole 4. Tato třída poskytuje kromě jiného i mechanismus pro takzvané *rampování* parametrů, což je proces, který eliminuje nežádoucí zvukové artefakty způsobené skokovou změnou některého z parametrů tím, že tento skok převede na pozvolný přechod z původní hodnoty na hodnotu novou (obr. 6.6).



Obrázek 6.6: Vizualizace rampování skokové změny parametru.

Existuje řada různých přístupů, kterými lze rampování parametrů realizovat. Jako příklad mohu uvést využití lineární interpolace či vyhlazovacího filtru prvního řádu s nekonečnou impulzní odezvou. Právě druhou z těchto dvou variant jsem se rozhodl použít i v rámci třídy `Module`.

Pro každý parametr jsou vytvořeny proměnné `current` a `target`, přičemž proměnná `current` obsahuje právě používanou hodnotu parametru a proměnná `target` hodnotu, na kterou má být parametr změněn. V metodě `processBlock` je pak během cyklu zpracování signálu vždy před zpracováním každého vzorku volána metoda `rampParameters`, jejímž úkolem je v případě, že se hodnoty obou proměnných neshodují, přepočítat hodnotu uloženou v proměnné `current` podle diferenční rovnice:

$$current = (target \cdot b) + (current \cdot a), \tag{6.1}$$

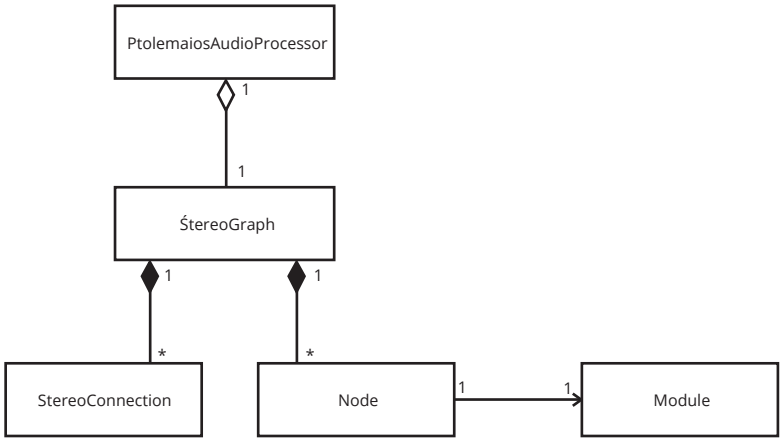
kde a a b jsou koeficienty vyhlazovacího filtru, vypočítané v metodě `prepareToPlay` na základě aktuální vzorkovací frekvence f_s a vhodně zvolené doby přechodu t pomocí rovnic:

$$\begin{aligned} a &= e^{-2\pi/(tf_s)}, \\ b &= 1 - a. \end{aligned} \tag{6.2}$$

6.3.2 Třída StereoGraph

Srdcem procesoru je třída `StereoGraph`, která reprezentuje strukturu orientovaného acyklického grafu podle definice 5.2. Tato třída uchovává pole objektů třídy `Node`, představujících uzly grafu, současně s informací o jejich vzájemném propojení v podobě pole objektů třídy `StereoConnection`.

Každý uzel je rozpoznatelný dle unikátního identifikačního čísla `nodeID`. Kromě něj spravuje třída `Node` také ukazatel na objekt třídy `Module` a dále množinu vlastností (typ, index a souřadnice) prostřednictvím objektu třídy `NamedValueSet`. Třída `StereoConnection` pak obsahuje identifikační čísla zdrojového a cílového uzlu.



Obrázek 6.7: Struktura třídy `StereoGraph` a její vztah s procesorem pluginu.

Pro každý uzel uchovává třída `StereoGraph` také objekt třídy `AudioBuffer` a díky třídě `PredecessorsTable` i množinu jeho předchůdců. Pro správný průchod grafem je k dispozici seznam `topolSorted`, jenž je topologicky uspořádán pomocí algoritmu 5.2. Nejdůležitější metody třídy `StereoGraph` jsou uvedeny v následujícím výčtu.

`addNode` přijímá jako parametr ukazatel na objekt třídy `Module` a volitelně i identifikační číslo, které pokud není zadáno, je automaticky vygenerováno. Metoda zkontroluje, zda-li se uzel s daným identifikačním číslem již v grafu nevyskytuje a v případě že ne, vytvoří nový objekt třídy `Node` společně s novým audio bufferem a řádkem tabulky předchůdců. Na závěr metoda zahrne uzel do seznamu `topolSorted` a zavolá metodu `prepareToPlay` objektu třídy `Module`.

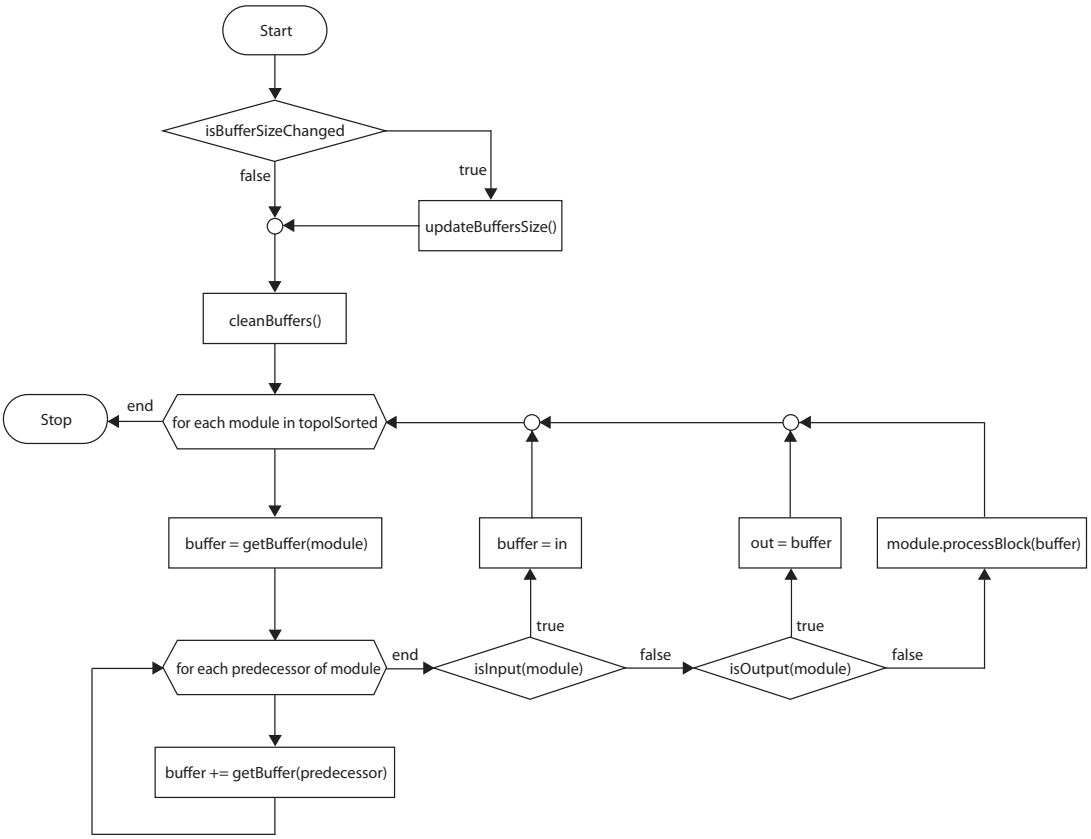
`removeNode` je metoda, jenž má za úkol odebrat z grafové struktury uzel s daným identifikačním číslem. Před samotným odebráním uzlu jsou odstraněna veškerá spojení vedoucí z uzlu nebo naopak do něj, přičemž po jejich odstranění je provedeno nezbytné přeuspořádání seznamu `topolSorted`. Následně je odstraněn uzel včetně audio bufferu a odpovídajícího řádku z tabulky předchůdců.

`addStereoConnection` je metoda, která prostřednictvím parametrů přijímá identifikační čísla zdrojového a cílového uzlu, pro něž vytvoří nový objekt třídy `StereoConnection` reprezentující jejich vzájemné propojení. Před samotným vytvořením metoda zkontroluje, zda-li jsou oba uzly s uvedenými identifikačními čísly v grafu obsaženy a jestli propojení mezi nimi již neexistuje. Po propojení uzlů pak metoda detekuje případný

vznik cyklu. Pokud byl v grafu cyklus nalezen, je propojení odstraněno a uživatel je prostřednictvím grafického uživatelského rozhraní na tuto skutečnost upozorněn. V opačném případě metoda přeuspořádá seznam `topolSorted` a přidá záznam do řádku cílového uzlu v tabulce předchůdců.

`removeSteroConnection` opět pomocí parametrů přijímá identifikační čísla zdrojového a cílového uzlu a následně vyhledá a odstraní jejich propojení. Po jeho odstranění je odebrán i záznam z řádku cílového uzlu v tabulce předchůdců a poté přeuspořádán seznam `topolSorted`.

`processBlock` je nejdůležitější metoda třídy `StereoGraph`, jejímž úkolem je zajistit správný průchod signálu, přes všechny uzly grafu. Metoda prostřednictvím parametru získává od procesoru referenci na objekt třídy `AudioSampleBuffer`, který obsahuje vstupní zvuková data. Na začátku metoda zkontroluje, zda-li nedošlo ke změně velikosti vstupního audio bufferu a pokud ano, změní velikost audio bufferů i všech uzlů grafu. Po této kontrole je původní obsah audio bufferů všech uzlů vynulován. Následně metoda postupně prochází seznamem `topolSorted`, přičemž pro každý v něm obsažený uzel provede dva kroky. Prvním krokem je naplnění audio bufferu uzlu součtem audio bufferů všech předchůdců tohoto uzlu. Druhým krokem je pak zpracování naplněného bufferu zvukovým modulem. Popsaný algoritmus je demonstrován pomocí vývojového diagramu zobrazeného na obrázku 6.8.



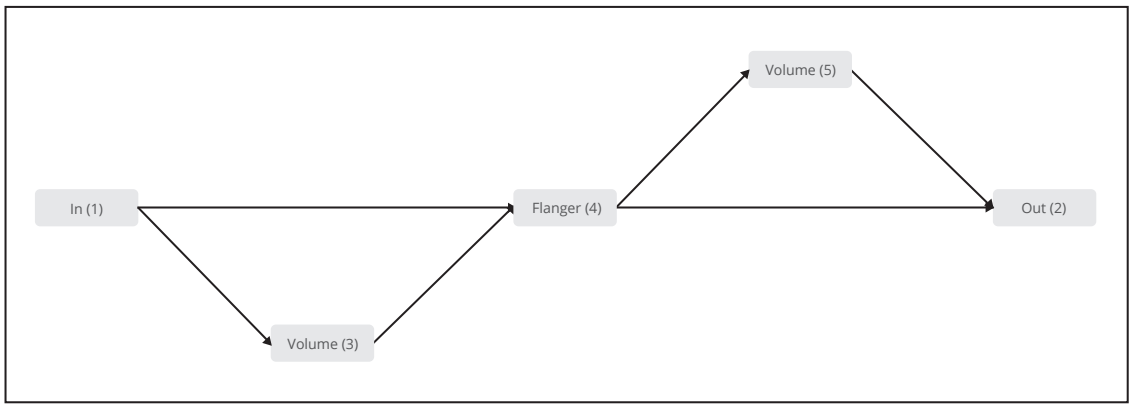
Obrázek 6.8: Vývojový diagram demonstrující postup metody `processBlock`.

6.3.3 Uložení a obnovení stavu pluginu

Během práce s projektem může být v případě potřeby hostitelskou aplikací volána metoda procesoru `getStateInformation`, jejímž úkolem je uložení vnitřního stavu pluginu. Pro účel serializace požadovaných dat jsem zvolil obecný značkovací jazyk XML. Vytvořená XML struktura je před uložením převedena metodou `copyXmlToBinary` do binární podoby. Pro zpětný převod z binární podoby do XML je využívána metoda `getXmlFromBinary`, která je volána metodou `setStateInformation` během obnovování stavu pluginu. Pro ilustraci přikládám zjednodušenou stavovou strukturu pluginu.

```
<PTOLEMAIOS>
  <SETTINGS name="Ptolemaios" version="1.0.2" />
  <CONSTANTS maxModuleVolume="4" maxModuleFlanger="1"/>
  <MODULES>
    <MODULE id="1" type="1" index="0" x="0.05" y="0.5" />
    <MODULE id="2" type="2" index="0" x="0.95" y="0.5" />
    <MODULE id="3" type="3" index="0" x="0.3" y="0.22" />
    <MODULE id="4" type="4" index="0" x="0.5" y="0.44" />
    <MODULE id="5" type="3" index="0" x="0.7" y="0.78" />
  </MODULES>
  <CONNECTIONS>
    <CONNECTION sourceModule="1" destModule="3" />
    <CONNECTION sourceModule="1" destModule="4" />
    <CONNECTION sourceModule="3" destModule="4" />
    <CONNECTION sourceModule="4" destModule="5" />
    <CONNECTION sourceModule="4" destModule="2" />
    <CONNECTION sourceModule="5" destModule="2" />
  </CONNECTIONS>
  <PARAMETERS count="9">
    <PARAMETER name="pVolumeGain0" value="0" index="0" type="0" />
    <PARAMETER name="pVolumeGain1" value="0" index="1" type="0" />
    <PARAMETER name="pFlangerBypass0" value="50" index="4" type="0" />
    <PARAMETER name="pFlangerDepth0" value="50" index="4" type="0" />
    <PARAMETER name="pFlangerFreq0" value="0.18" index="5" type="0" />
    <PARAMETER name="pFlangerFeedback0" value="50" index="6" type="0" />
    <PARAMETER name="pFlangerType0" value="0" index="7" type="2" />
    <PARAMETER name="pFlangerQuad0" value="1" index="8" type="1" />
    <PARAMETER name="pFlangerIntensity0" value="1" index="8" type="1" />
  </PARAMETERS>
</PTOLEMAIOS>
```

Značka `SETTINGS` definuje název a verzi pluginu, pro který je struktura určena a společně s atributy uloženými pod značkou `CONSTANTS` slouží k ověření kompatibility při obnově stavu pluginu. Značka `MODULES` obsahuje seznam všech použitých modulů, včetně informací o jejich typu a pozici. Propojení mezi moduly je reprezentováno značkami `CONNECTION`. Značka `PARAMETERS` obsahuje výčet všech parametrů pluginu, přičemž u každého je evidován identifikační název, hodnota, index a typ. Všechny zmíněné značky jsou zahrnuty pod kořenovou značkou `PTOLEMAIOS`. Výše uvedený příklad stavové struktury pluginu odpovídá zapojení modulů zobrazeném na obrázku 6.9.



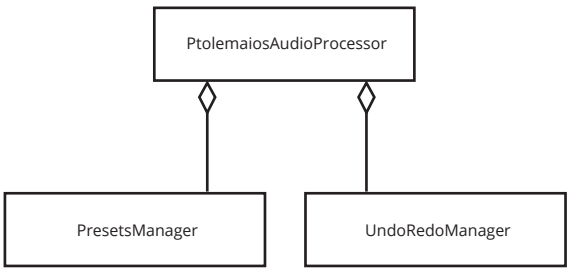
Obrázek 6.9: Ukázka jednoduchého zapojení modulů.

6.3.4 Třída PresetsManager

S ukládáním stavu pluginu úzce souvisí třída `PresetsManager`, jíž procesor pluginu také disponuje. Tato třída uživateli umožňuje ukládat stav pluginu do souboru, případně stav pluginu z dříve uloženého souboru načítat. Při spuštění pluginu je procesorem volána metoda `importPresets`, která prohledá složku `Ptolemaios` umístěnou ve složce `TNT`, jež se nachází ve složce `Dokumenty`. Jejím úkolem je v uvedené složce nalézt všechny soubory s příponou `tnt` a uložit jejich výčet do interní stromové datové struktury. Samotné načítání a ukládání souboru je prováděno voláním metod `loadPreset` a `savePreset`. Za zmínku také stojí metoda `restoreFactoryPresets`, která po jejím zavolání vytvoří ve složce `Ptolemaios` soubory obsahující připravená přednastavení pluginu.

6.3.5 Třída UndoRedoManager

Praktickým rozšířením pluginu bylo přidání třídy `UndoRedoManager`, která uchovává historii změn provedených v rámci objektu třídy `StereoGraph`. Po každé provedené změně (přidání uzlu, odebrání propojení, apod.) je stav pluginu uložen voláním metody `saveState` do interního seznamu `states` jako objekt třídy `XmlElement`. Procházení historie změn je následně možné voláním metod `undo` a `redo`. Pokud se plugin nachází v některém stavu historie (je možný krok vpřed) a dojde k nové změně, je původní posloupnost dopředných kroků smazána a nahrazena stavem uloženým právě po této změně. Vylepšením této třídy by bylo definovat pro každou možnou akci odpovídající protiakci, čímž by se namísto posloupnosti stavů pluginu ukládala pouze posloupnost provedených akcí.

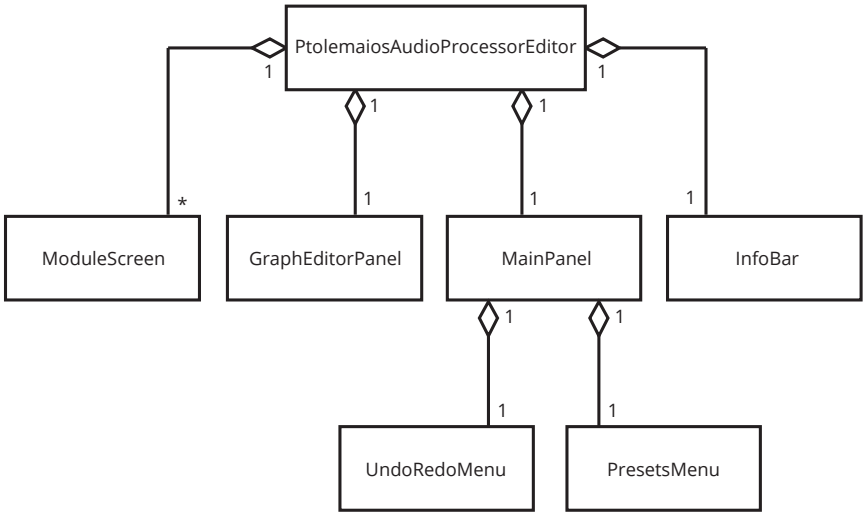


Obrázek 6.10: Vztah procesoru s třídami `UndoRedoManager` a `PresetsManager`.

6.4 Editor pluginu

Jak už bylo uvedeno dříve, editor obsahuje prvky, jenž společně tvoří grafické uživatelské rozhraní pluginu. Základním stavebním kamenem, ze kterého jsou odvozeny všechny grafické prvky, je takzvaná komponenta, jež je reprezentována třídou `Component`. Jednotlivé komponenty lze do sebe prostřednictvím metod `addChildComponent` a `addAndSetVisible` zanořovat, čímž je vytvářena stromová hierarchie komponent, jejímž kořenem je v případě pluginu právě jeho editor.

Mezi nejdůležitější metody třídy `Component` patří také metoda `paint`, které je prostřednictvím parametru předána reference na objekt třídy `Graphics`; ten představuje grafický kontext, do kterého metoda `paint` vykresluje svůj grafický obsah. Velikost komponenty lze nastavit pomocí metody `setSize`. Metoda `setBounds` umožňuje kromě velikosti komponenty nastavit i její pozici v rámci komponenty nadřazené. Po změně velikosti voláním jedné ze jmenovaných metod je automaticky volána metoda `resized`, v níž je dané komponentě umožněno přenastavit velikost svých podřazených komponent.

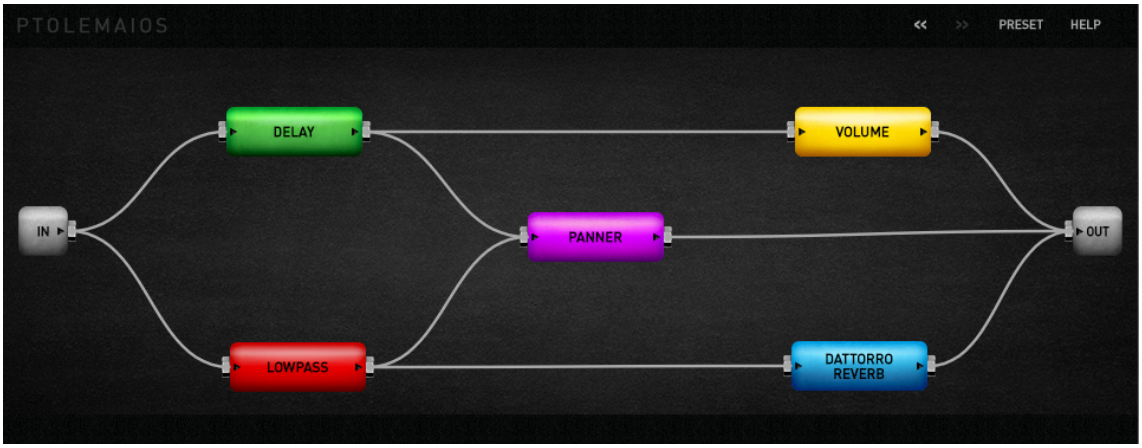


Obrázek 6.11: Diagram struktury editoru.

Na obrázku 6.11 je zobrazena struktura editoru, která odpovídá návrhu ovládací obrazovky z kapitoly 3. Třída `MainPanel` zapouzdřuje třídy `PresetsMenu` a `UndoRedoMenu`, s jejichž využitím poskytuje rozhraní pro načítání a ukládání presetů (rozbalovací nabídka *presets*) a procházení historie změn (tlačítka *zpět* a *vpřed*). Pro výpis informačních a chybových zpráv je určena třída `InfoBar`, konkrétně pak její metoda `printMessage`.

Prostřednictvím třídy `GraphEditorPanel` je uživateli umožněno vkládat zvukové moduly a vzájemně je tažením myši propojovat. Jednotlivé moduly jsou reprezentovány objekty třídy `ModuleNode`, jejich propojení pak objekty třídy `ModuleConnector`. Po kliknutí pravým tlačítkem myši do prostoru ovládacího panelu je uživateli zobrazena kontextová nabídka, která kromě vkládání modulů umožňuje i všechny aktuálně vložené instance modulů smazat, případně pouze zrušit všechna jejich propojení. Třída `ModuleScreen` slouží jako abstraktní třída, z níž jsou odvozeny všechny třídy reprezentující ovládací obrazovky konkrétních typů modulů. Podrobnějšímu popisu třídy `ModuleScreen` a tříd z ní odvozených je věnována sekce 6.4.1.

Od svého prvního návrhu prošel vzhled editoru celou řadou iterací, ve kterých byly kromě jiného promítnuty i požadavky získané během průběžného uživatelského testování. Jedním z důležitých rozšíření původního návrhu, bylo přidání ovládacího prvku *mini panel*, díky němuž je uživateli umožněno řídit intenzitu libovolného zvukového modulu přímo z hlavní obrazovky editoru. *Mini panel* je možné pro každý vložený modul zobrazit či skrýt pomocí pravého tlačítka myši. Přestože vzhled editoru pluginu vychází z paradigmatu *skeumorfismu*, snažil jsem se o zachování co možná největší jednoduchosti a přehlednosti. Finální vzhled hlavní obrazovky pluginu si můžete prohlédnout na obrázku 6.12.



Obrázek 6.12: Finální vzhled hlavní obrazovky pluginu.

Pro přenos informací z procesoru pluginu do jeho editoru se používá dvou přístupů. Prvním z nich je periodické dotazování ze strany editoru, čehož je využito například ke kontrole, zda nedošlo ke změně některého z parametrů procesoru hostitelskou aplikací; tudíž jestli není zapotřebí aktualizovat hodnotu některého z ovládacích prvků. Editor pluginu dědí ze třídy `Timer`, přičemž ve svém konstruktoru voláním metody `startTimerHz` spouští časovač, jenž dle nastavené frekvence pravidelně způsobuje vyvolání metody `timerCallback`.

Druhý přístup je používán v případě, kdy je zapotřebí přenést informace ze strany procesoru. Jelikož procesor nedisponuje ukazatelem na editor a nemůže tedy předat informace editoru přímo, je využíváno systému *broadcasteru* a *listenerů*. Z tohoto důvodu procesor pluginu dědí ze třídy `ChangeBroadcaster` a editor pluginu ze třídy `ChangeListener`. Editor se v rámci svého konstruktoru registruje pomocí metody procesoru `addChangeListener` jako *listener*. V případě potřeby pak procesor asynchronně volá metodu `sendChangeMessage`, která způsobí vyvolání metody editoru `changeListenerCallback`. Ve svém destruktoru se editor na závěr voláním metody procesoru `removeChangeListener` odregistruje.

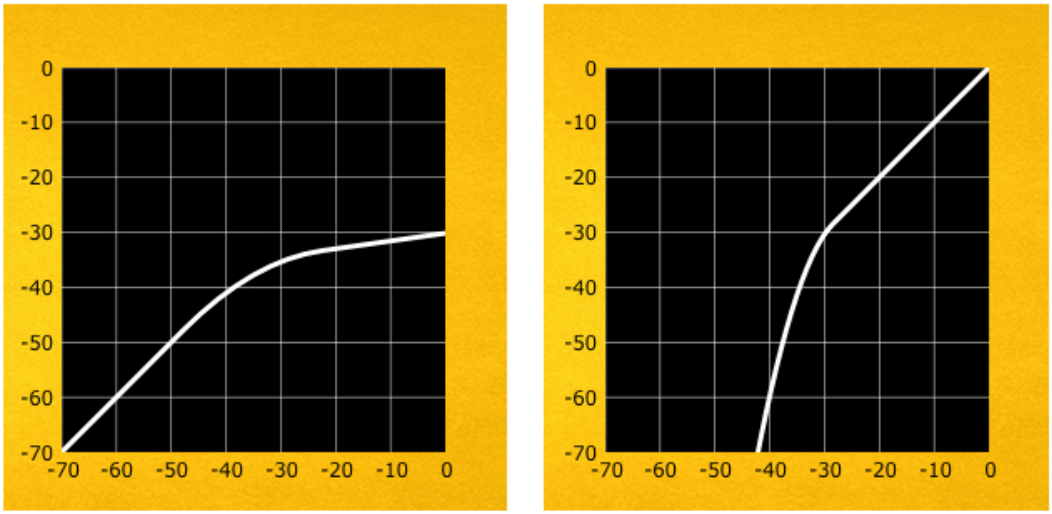
6.4.1 Třída ModuleScreen

Jak již bylo naznačeno výše, pro každý typ zvukového modulu je k dispozici ovládací obrazovka, jež je reprezentována konkrétní třídou odvozenou od abstraktní třídy `ModuleScreen`. Na těchto obrazovkách jsou rozmístěny ovládací prvky, pomocí kterých je uživateli umožněno nastavovat jednotlivé parametry procesoru.

Pro pohodlné nastavování parametrů třídy `AudioParameterFloat` byla odvozením od třídy `Slider` vytvořena třída `AnimatedKnob`. Třída `AnimatedKnob` přijímá pomocí konstruktoru objekt třídy `Image`, jenž obsahuje takzvaný *strip*, což je obrázek skládající se ze všech možných pozic knobu ležících vertikálně pod sebou. Podle aktuálně nastavené

hodnoty parametru se pak vykreslí pouze odpovídající část stripu. Pokud uživatel začne pohybovat s knobem, je zavolána metoda obrazovky `sliderDragStarted`, která prostřednictvím metody parametru `beginChangeGesture` informuje hostitelskou aplikaci o začátku změny parametru. Samotné nastavení hodnoty daného parametru poté probíhá pomocí metody obrazovky `sliderValueChanged`. Po uvolnění knobu je volána metoda obrazovky `sliderDragEnd`, v níž je hostitelská aplikace metodou parametru `endChangeGesture` informována o ukončení změny parametru.

Přepínače hodnoty parametrů třídy `AudioParameterBool` jsou realizovány pomocí třídy `ImageButton` odvozené od třídy `Button`. Po stisknutí konkrétního přepínače je volána metoda obrazovky `buttonClicked`, v níž je hodnota daného parametru nastavena. Pro nastavení hodnoty parametrů třídy `AudioParameterChoice` je využita třída `ComboBox`, která uživateli poskytuje rozbalovací nabídku s možnostmi. Po výběru některé z možností je volána metoda obrazovky `comboBoxChanged`, jejímž úkolem je nastavit hodnotu parametru právě dle vybrané možnosti.



Obrázek 6.13: Vizualizace dynamických křivek kompresoru a expanderu.

Kromě ovládacích prvků disponují některé obrazovky i prvky vizualizačními. První z nich je měřič hlasitosti reprezentovaný třídou `AnimatedMeter`, která podobně jako třída `AnimatedKnob` přijímá prostřednictvím parametru konstruktoru obrázkový strip. Podle hodnoty předávané metodě `displayValue` je pak zobrazována odpovídající část stripu. Dalšími vizualizačními prvky jsou třídy `CompressorVisualisation` a `ExpanderVisualisation`; ty umožňují vykreslení dynamické křivky kompresoru a expanderu na základě nastavení jejich parametrů *threshold* a *ratio*. Posledními implementovanými vizualizačními prvky, jež slouží pro zobrazení ekvalizační křivky filtrů a ekvalizéru, jsou třídy `FilterVisualisation` a `EqualizerVisualisation`.

Každá třída odvozená ze třídy `ModuleScreen`, jenž reprezentuje obrazovku konkrétního typu modulu, musí implementovat metody `updateMeters` a `updateParameters`. První z těchto dvou metod slouží k aktualizaci měřičů hlasitosti, druhá pak ke kontrole, zda-li nedošlo ke změně některého z parametrů modulu jinak než prostřednictvím ovládacího prvku. Pokud ke změně došlo, je zapotřebí tuto změnu promítnou i na odpovídající ovládací prvek. Obě zmíněné metody jsou periodicky volány editorem pluginu. Ovládací a vizualizační prvky, jež byly popsány v této sekci, si čtenář může prohlédnout v příloze C.

Kapitola 7

Testování

Tato kapitola je věnovaná testování pluginu *Ptolemaios*. Testování jsem rozdělil do dvou sekcí. V první z nich popisuji nástroje, zvukové signály a technické vybavení, které jsem využil pro testování funkčnosti pluginu. Druhá sekce se pak zabývá uživatelským testováním.

7.1 Testování funkčnosti

Plugin byl implementován v programovacím jazyce C++ prostřednictvím vývojového prostředí *Microsoft Visual Studio Community 2015*, které je volně ke stažení z oficiálních stránek programu[1]. Toto vývojové prostřední poskytuje kromě jiného i sadu pokročilých testovacích a ladících nástrojů, které byly během vývoje pluginu často používány. Visual Studio disponuje možností připojit se k procesu hostitelské aplikace (položka *Attach to Process* v menu *Debug*) a ladit tak plugin za běhu například pomocí krokování programu či hlídání změny hodnoty vybrané proměnné. Díky tomuto postupu se podařilo vyřešit řadu problému vzniklých odlišným chováním různých hostitelských aplikací. Plugin byl testován na osobních počítačích s operačními systémy *Microsoft Windows* (verze 7, 8.1 a 10) v těchto hostitelských aplikacích:

Steinberg Cubase 5	*Image-Line FL Studio 9*
Steinberg Cubase Elements 7	*Image-Line FL Studio 10*
Steinberg Cubase Pro 8.5	*Image-Line FL Studio 11*
Steinberg VST3 Plugin Test Host	*Image-Line FL Studio 12*
Steinberg WaveLab 6	*Cakewalk Sonar Platinum*
Steinberg Nuendo 4	*Juce Audio Plugin Host*
Avid Sibelius 8	*Tobybear MiniHost 1.64*
Ableton Live 9	*VST Plugin Analyser*
n-Track Studio 7	*PreSonus Studio One 3*

Kromě testování pluginu jako celku byla důkladně testována i funkčnost jeho jednotlivých zvukových modulů. K tomuto účelu jsem sestavil početnou množinu testovacích zvukových signálu. Tato množina obsahuje zvukové stopy zpěvu, rapu a hudebních nástrojů,

[1] https://www.visualstudio.com

ale také stopy jednoduchých sinusových a šumových signálů. Všechny testovací signály jsou uloženy na přiloženém DVD ve formátu *wav* se vzorkovací frekvencí 44100 Hz, bitovou hloubkou 16 bitů a jedním či dvěma kanály. Signály jsem rozřadil dle jejich typu do adresářů, které jsou včetně počtu do nich náležících signálů uvedeny v následujícím výčtu:

21x bass	*27x guitar*	*25x piano*	*10x sine*
21x brass	*19x kick*	*14x sax*	*6x noise*
10x clap	*13x lead*	*21x snare*	*24x rap*
10x drums	*17x organ*	*14x strings*	*14x singing*
11x flute	*4x pad*	*14x synth*	*1x speech*

Pro analýzu zpracovávaného signálu jsem během testování pluginu v hostitelských aplikacích využíval analytické nástroje v podobě zásuvných modulů třetích stran. Za zmínku stojí především frekvenční analyzátory *Voxengo SPAN*[2] a *Blue Cat's FreqAnalyst*[3], které oba pracují na principu *rychlé Fourierovy transformace*. Pomocí těchto analyzátorů a bílého či růžového šumu jsem mohl sledovat frekvenční odezvu testovaného systému. Užitečným byl také měřící nástroj *iZotope Insight*[4].

Obrázek 7.1: Frekvenční analyzátor *Voxengo SPAN*.

Velmi často jsem k ověření správné funkčnosti pluginu využíval také funkcí hostitelské aplikace *VST Plugin Analyser*[5]. Tato aplikace disponuje sadou testů, generátorem bílého a růžového šumu a nástroji pro měření dynamické a frekvenční charakteristiky. Ukázky některých naměřených charakteristik jsou umístěny v příloze D.

[2] http://www.voxengo.com/product/span
[3] https://www.bluecataudio.com/Products/Product_FreqAnalyst
[4] https://www.izotope.com/en/products/mix/insight.html
[5] http://www.pcjv.de/applications/measurement-programs/vst-plug-in-analyser-2-0/

Co se týká hardwarové výbavy, byl pro testování nejčastěji využíván osobní počítač s operační pamětí o kapacitě 16 GB a tříjádrovým procesorem řady *AMD Athlon II X3* o frekvenci 3000 MHz v kombinaci s externími zvukovými kartami *M-Audio MobilePre mkII*, *Steinberg UR242*, *TC Electronic Konnekt 8* a *TC Electronic Impact Twin*. Pro poslech zpracovávaného signálu jsem měl k dipozici tyto studiová sluchátka a poslechové monitory:

KRK VXT 6	*AKG K-712 Pro*	*AKG K-171 MKII*
KRK Rokit 6 G2	*AKG K-121 Studio*	*Beats Studio*
JBL LSR305	*AKG K-271 MKII*	*Beats Wireless*

7.2 Uživatelské testování

Vývoj pluginu probíhal v iteracích, přičemž na konci každé iterace jsem aktuálně dokončenou verzi poskytl několika testerům. Případné postřehy a opravy chyb jsem následně zahrnul společně s novou funkcionalitou do verze další. Pro získání zpětné vazby z reálného nasazení pluginu jsem k závěru vývoje provedl uživatelské testování.

Skupinu uživatelů jsem krátce seznámil s funkcemi a ovládáním pluginu. V této skupině byli jak lidé, kteří se postprodukcí hudby zabývají krátce na amatérské úrovni, tak studioví inženýři a hudební producenti, kteří se postprodukcí již řadu let živí. Všichni uživatelé pak měli několik dní na bližší seznámení s pluginem, po jejichž uplynutí byli požádání o vyplnění krátkého dotazníku, který jsem pro tento účel sestavil. Tento dotazník skládající se ze třinácti otázek je uveden v příloze A.1.

Úvodní tři otázky se týkají zkušenostmi uživatele s nahráváním a postprodukcí hudby, zbývající pak mírou spokojenosti s provedením a praktickým využitím pluginu *Ptolemaios*. Kromě zodpovězení otázek mohli uživatelé uvést, ve kterých hostitelských aplikacích plugin vyzkoušeli a slovně shrnout silné a slabé stránky pluginu. Někteří ze zkušenějších uživatelů byli také požádání o sepsání krátkého posudku. Získané posudky jsou uvedeny v příloze A.2. Výsledky uživatelského testování demonstruje tabulka 7.1.

	Rozsah	**Průměrná známka**	**Směrodatná odchylka**
Zvuková kvalita	1-5	1.5	0.63
Využití pro postprodukci	1-5	1.36	0.48
Celkový dojem	1-5	1.43	0.62

Tabulka 7.1: Výsledky uživatelského testování.

Z výsledků uživatelského testování vyplývá, že plugin *Ptolemaios* lze úspěšně využít pro postprodukci populární hudby. Všechny testující uživatele příjemně zaujal vzhled pluginu, všichni také považují výběr zvukových efektů a procesorů za vyhovující a dostatečný. Většina z testujících označuje ovládání pluginu jako jednoduché, přehledné a intuitivní; objevil se však i požadavek o nalezení rychlejšího způsobu propojování modulů.

Po zvukové stránce nejvíce úspěchu sklidily dozvukové procesory. Převážná většina uživatelů by ocenila rozšíření systému o frekvenční analyzátory signálu a interaktivní grafy, pročež tímto směrem pravděpodobně povede budoucí vývoj. Jak se předpokládalo, zejména méně zkušené uživatele potěšila nabídka zabudovaných přednastavení, nicméně někteří z testujících by uvítali jejich vyšší počet.

Uživatelské testování se ukázalo jako velice přínosné. Některé z uživateli navržených změn byly již do systému začleněny, ostatní budou předmětem dalšího vývoje pluginu.

Kapitola 8

Závěr a budoucí práce

8.1 Shrnutí

V rámci této diplomové práce jsem navrhl a realizoval komplexní systém určený pro postprodukci populární hudby. Tento systém má podobu *VST* pluginu a byl implementován v programovacím jazyce C++ s využitím aplikačního frameworku *JUCE*. Plugin nabízí bohatou kolekci zvukových efektů a procesorů, jejichž instance si uživatel může zapojit do libovolné acyklické grafové struktury. Uživateli je tedy umožněno si jednoduše a rychle vytvářet vlastní originální zvukové multiefekty. Plugin disponuje vestavěnou databází přednastavení a umožňuje uživateli ukládat a načítat přednastavení vlastní. Pro tento účel jsem využil značkovacího jazyku *XML*. Plugin uchovává historii provedených změn, v níž uživatel může procházet pomocí tlačítek *zpět* a *vpřed*.

Parametry každého z implementovaných zvukových efektů a procesorů lze detailně nastavit prostřednictvím ovládacích prvků umístěných na vlastních obrazovkách. Některé z těchto obrazovek kromě ovládacích prvků obsahují i grafické prvky vizualizující nastavení konkrétních parametrů. Hlasitost výstupního signálu může uživatel sledovat pomocí měřičů hlasitosti. Grafické uživatelské rozhraní bylo realizováno s ohledem na přehlednost, jednoduchost a intuitivnost. K testování funkčnosti pluginu i jeho jednotlivých modulů byla sestavena množina testovacích signálů. Pro získání zpětně vazby z reálného nasazení pluginu jsem provedl uživatelské testování, z jehož výsledků vyplývá, že plugin *Ptolemaios* je úspěšně využitelný pro postprodukci populární hudby.

8.2 Budoucí práce

Z uživatelského testování vzešlo několik podnětných nápadů na rozšíření či vylepší stávající funkčnosti pluginu, přičemž některé z těchto postřehů byly již do systému začleněny. Příkladem realizovaných rozšíření je zvýšení počtu a změna kategorií vestavěných přednastavení či doplnění ovládacího prvku *mini panel*, který umožňuje jednoduše měnit intenzitu libovolného vloženého modulu z hlavní ovládací obrazovky pluginu.

V krátkém časovém horizontu bych chtěl doplnit moduly pro dynamické procesory *gate*, *de-esser* a *limitér*. Dále bych chtěl vytvořit multipásmové varianty některých modulů. Pro rozšíření cílové skupiny uživatelů by bylo vhodné plugin převést i do formátů *Audio Unit* a *AAX*. V delším časovém horizontu bych rád obohatil uživatelské rozhraní o interaktivní grafy a pokročilé nástroje pro analýzu zvukového signálu.

Literatura

[1] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; aj.: *Introduction to Algorithms*. Prentice Hall India Learning Private Limited, třetí vydání, 2010, ISBN 978-8120340077, 1292 s.

[2] Dattorro, J.: Effect Design* Part 1: Reverberator and Other Filters. *Journal of the Audio Engineering Society*, ročník 45, č. 9, Září 1997.

[3] Dattorro, J.: Effect Design* Part 2: Delay-Line Modulation and Chorus. *Journal of the Audio Engineering Society*, ročník 45, č. 10, Říjen 1997.

[4] Dattorro, J.: Effect Design* Part 3 Oscillators: Sinusoidal and Pseudonoise. *Journal of the Audio Engineering Society*, ročník 50, č. 3, Březen 2002.

[5] Dvořák, J.: Úlohy teorie grafů. Ústav automatizace a informatiky, Fakulta strojního inženýrství, Vysoké učení technické v Brně, Brno, 2000.

[6] Černý, J.: Základní grafové algoritmy. Katedra matematiky, Matematicko-fyzikální fakulta, Univerzita Karlova, Praha, 2010.

[7] Goel, A.: Detect Cycle in a direct graph using colors [online]. Dostupné z: http://www.geeksforgeeks.org/detect-cycle-direct-graph-using-colors, 2016 [cit. 12. 12. 2016].

[8] Guerin, R.: *MIDI Power!* Thomson Course Technology, druhé vydání, 2005, ISBN 9781598630855, 410 s.

[9] Lyons, R.: Digital Envelope Detection: The Good, the Bad, and the Ugly [online]. Dostupné z: https://www.dsprelated.com/showarticle/938.php, Duben 2016 [cit. 8. 4. 2017].

[10] Mac Dev Center: Audio Unit Programming Guide [online]. Dostupné z: https://developer.apple.com/library/content/documentation/MusicAudio/Conceptual/AudioUnitProgrammingGuide/Introduction/Introduction.html, 2014 [cit. 5. 1. 2017].

[11] Mach, V.: *Akustická měření v reálném prostředí*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Brno, 2009.

[12] Moorer, J. A.: About This Reverberation Business. *Computer Music Journal*, ročník 3, č. 2, Červen 1979.

[13] Orfanidis, S.: *Introduction to Signal Processing*. Prentice Hall, Srpen 1995, ISBN 978-0132091725, 798 s.

[14] Pirkle, W.: *Designing Audio Effect Plug-Ins in C++*. Focal Press, první vydání, 2012, ISBN 978-0240825151, 560 s.

[15] Pirkle, W.: *Designing Software Synthesizer Plug-Ins in C++*. Focal Press, 2014, ISBN 978-1138787070, 760 s.

[16] Redmon, N.: EarLevel Engineering [online]. Dostupné z: http://www.earlevel.com, 2014 [cit. 17. 2. 2017].

[17] Reiss, J. D.; McPherson, A.: *Audio Effects*. CRC Press, 2014, ISBN 9781466560284.

[18] Robinson, M.: *Getting Started with JUCE*. Packt Publishing Ltd, 2013, ISBN 9781783283323, 158 s.

[19] Schimmel, J.: Studiová a hudební elektronika. Fakulta elektrotechniky a komunikačních technologií, Vysoké učení technické v Brně, Brno, 2015.

[20] Schroeder, M. R.: Natural Sounding Artificial Reverberation. *Journal of the Audio Engineering Society*, ročník 10, č. 3, Červenec 1962.

[21] Sedláček, J.: O konečných orientovaných grafech. *Časopis pro pěstování matematiky*, ročník 82, č. 2, 1957: s. 192–215.

[22] Senior, M.: *Mixujeme hudbu v domácím studiu*. Brno: Computer Press, 2014, ISBN 978-80-251-3798-7, 360 s.

[23] Smith, J. O.: Comb Filters [online]. Dostupné z: https://ccrma.stanford.edu/~jos/pasp/Comb_Filters.html, 2012 [cit. 6.5.2017].

[24] Steinberg Media Technologies GmbH: ASIO 2.3 SDK Specification. Stažitelné z: https://www.steinberg.net/en/company/developers.html, 2013 [cit. 10. 4. 2017].

[25] Steinberg Media Technologies GmbH: VST 3 SDK Documentation. Stažitelné z: https://www.steinberg.net/en/company/developers.html, 2017 [cit. 11. 4. 2017].

[26] Talbot-Smith, M.: *Audio Engineer's Reference Book*. Focal Press, druhé vydání, Červenec 2001, ISBN 978-0240516851, 676 s.

[27] Tichý, V.: *Digitální zvukový efekt typu reverb využívající konvoluci signálu s impulsní charakteristikou poslechového prostoru*. Diplomová práce, Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Brno, 2009.

[28] Trkal, T.: *Úprava vokálních stop v SW postprodukci hudby*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, Brno, 2014.

[29] Večerka, A.: Grafy a grafové algoritmy. Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého, Olomouc, 2007.

[30] Vlachý, V.: *Praxe zvukové techniky*. Praha: Muzikus, třetí vydání, 2008, ISBN 978-80-86253-46-5, 298 s.

[31] Zölzer, U.: *Digital Audio Signal Processing*. John Wiley & Sons, 1997, ISBN 0471972266.

[32] Zölzer, U.; Amatriain, X.; Arfib, D.; aj.: *DAFX - Digital Audio Effects*. John Wiley & Sons, 2002, ISBN 9780471490784.

Přílohy

Příloha A

Uživatelské hodnocení aplikace

A.1 Dotazník

Pro získání zpětné vazby od uživatelů, kterým jsem poskytl plugin k testování, jsem sestavil tento dotazník. První tři otázky jsou zaměřeny na zkušenosti uživatelů s postprodukcí hudby a lze na ně odpovědět buďto *ano* nebo *ne*.

- Máte zkušenosti s nahráváním hudebních skladeb?
- Zabýváte se postprodukcí hudebních nahrávek?
- Setkal/a jste se dříve s podobným typem pluginu?

Dalších sedm otázek se týká spokojenosti s provedením aplikace. Cílem těchto otázek je získání potřebných informací pro budoucí inovace. Odpovědi na tuto sadu otázek jsou *ano*, *spíše ano*, *nevím*, *spíše ne* a *ne*.

- Jste spokojený/á se vzhledem aplikace?
- Přijde Vám ovládání aplikace intuitivní?
- Považujete výběr zvukových procesorů a efektů za dostačující?
- Vyhovuje Vám rozmístění ovládacích a vizualizačních prvků?
- Uvítal/a byste rozšíření pluginu o analyzátory signálu?
- Vyhovuje Vám množství přednastavených presetů?
- Považujete sadu presetů za vyhovující pro dané účely?

Poslední tři otázky hodnotí celkovou použitelnost aplikace při postprodukci hudby a lze je oznámkovat známkami 1-5 dle české školní stupnice, tedy jednička znamená nejlepší hodnocení, pětka naopak nejhorší.

- Jak byste hodnotil/a plugin z hlediska zvukové kvality?
- Je dle Vašeho názoru aplikace využitelná pro profesionální postprodukci hudby?
- Jak byste ohodnotil/a Váš celkový dojem z pluginu Ptolemaios?

A.2 Hodnocení

Jan Bičan

– audio inženýr, hudební skladatel, muzikant

Plugin Ptolemaios lze považovat za univerzální prostředek pro zpracování zvuku. Počet modulů a měnitelných parametrů je dostačující, zároveň je ale zachována celková přehlednost ovládání, které je doplněno o užitečný ovládací prvek mini panel. Praktická jsou také tlačítka pro procházení historie změn. Z nabídky modulů mne po zvukové stránce nejvíce zaujal modul Dattorro Reverb. Pokud bych měl pluginu něco vytknout, vidím nedostatky v třídění a počtu nabízených presetů. Celkově však zásuvný modul hodnotím velmi kladně.

Ondřej Žatkuliak

– audio inženýr, majitel nahrávacího studia, hudební producent

Ptolemaios je svou versatilitou a varibilitou velmi originální a mnohostranný plugin. Díky množství efektových modulů a možnosti jejich libovolného pořadí je použitelný téměř v jakékoli situaci. Osobně mě velice zaujal převážně po zvukové stránce zdařilý modul reverbu, který v mém studiu už používáme v praxi.

Štěpán Bárta

– audio inženýr, majitel nahrávacího studia, hudební producent

Ptolemaios mě překvapil hned po prvním otevření svým provedením, respektive způsobem, kterým se v něm dá pracovat. Tento sytém zásuvných modulů, které se dají vzájemně propojovat, kombinovat a větvit je velice ojedinělý a doteď jsem ho viděl pouze v profesionálních programech pro post-produkci videa. Protože jsem si tento způsob práce u videa oblíbil, byl mi hned ze začátku velice sympatický i u Ptolemaia a hned jsem si na něj zvykl. Jistě, spousta pluginů umí podobně simulovat rack, tedy sériové zapojení, ale audio plugin, u kterého by se dalo vytvořit i poměrně složité paralelní zapojení modulů, jsem ještě neviděl.

Svým způsobem Ptolemaios kombinuje audio plugin, který simuluje rack a zároveň nahrazuje i sendy, což se může v některých DAWech hodit například právě proto, že v nich je posílání do sendů zbytečně složité. Mluvím například o Pro Tools, kde se groupy i sendy řeší přes AUX track a virtuální výstupy a vstupy (pokud přebíráte po někom sešnu, je prakticky nemožné se v tom vyznat, obzvlášť, pokud člověk před vámi každou stopu, vstup a výstup pečlivě nepojmenuje) nebo o FL studiu, kde je systém sendů řešen klasickou audio stopou a virtuálním kabelem, který do ní vede. Po otevření Ptolemaia je vždy naprosto jasné, kam jaký výstup vede a jak daný proces funguje bez zdlouhavého rolování, zasólovávání a mutování.

Myslím, že se Ptolemaios uplatní i ve střižnách, které podporují pluginy třetích stran (například Sony Vegas, Davinci Resolve apod), třeba k mixu propagačních videí nebo rozhovorů, kde se kombinují stopy hudby, voiceoveru a kontaktního zvuku z placu. Pokud střihačům odpadne nutnost exportu do OMF a následného soundmixu v jiném softwaru, ušetří spoustu času. Pluginy, které jsou dodávány s těmito střižnami rozhodně ani nedosahují takové zvukové kvality a jednoduchosti ovládání, které nabízí Ptolemaios.

Vzhled Ptolemaia mi přijde vkusný, mám rád jednoduchost a přehlednost. Líbí se mi možnost vytváření presetů a jejich následné vyvolání i možnost použití už hotových presetů.

V ovládání mi trochu chybí možnost vypnutí a zapnutí jednotlivých modulů bez nutnosti jejich otevření a popřípadě možnost nastavení poměru wet-dry nebo drive. To by zrychlilo workflow při práci s Ptolemailem na dvojnásobek. Co se týče kvality zvuku jednotlivých modulů, hodně mě překvapily filtry, ekvalizér a reverb, při testování se mi párkrát povedlo nakroutit hezčí zvuk s Ptolemaiem než s drahými pluginy, které běžně používám. Výběr modulů bych osobně rozšířil o de-esser.

Pokud bych měl shrnout vše výše uvedené, Ptolemaios má podle mě uplatnění od jednoduchého nazvučení demíčka až po mix voiceoverů nebo například stopy vokálu v moderním popovém songu, kde se pracuje hlavně s hally a delayi, které jsou dále prohnané přes různé filtry. Obzvlášť se ale hodí jako náhražka složité míchací konzole v programech, které nejsou primárně DAW a podporují VST, např. střižny apod.

Tomáš Lašan

– interpret, majitel nahrávacího studia

Plugin Ptolemaios je výkonný a efektivní nástroj, který dokáže pomoci jak u demo verzí určených pro poslech a kontrolu nahrané skladby, tak i u finálního mixu. Z nabídky modulů si lze vybrat základní dynamické, frekvenční i prostorové efekty, které potřebujete ke sjednocení zvuků v nahrávce. Kompletní nabídka zvukových efektů je však jen jedna z výhod programu. Začátečník i profesionál jistě ocení také jednoduché a intuitivní ovládání a možnost propojování jednotlivých efektů. Pokud hledáte konkrétní úpravu například pro piano, kytaru nebo jiné nástroje, pomůže vám bohatá nabídka přednastavených presetů. V mém nahrávacím studiu je tento plugin již zaběhnutým pomocníkem.

Václav Štirský

– audio inženýr, majitel nahrávacího studia, hudební skladatel

Jednoduché grafické prostředí, které je velmi intuitivní a rychle obslužné. Líbí se mi nápad propojení pomocí tzv. kabelů, kde i méně zkušený zvukový mistr může rychle a snadno porovnávat zapojení jednotlivých modulů a najít tak správné řešení pro danou situaci. Mile mě překvapily reverby, od kterých jsem ze začátku nic nečekal, ale po krátkém vyzkoušení jsem změnil názor. Snadné nastavení není přítěží při rychlé práci s hudebníky (zákazníky) za zády. Určitě se neobejdete bez dalších úprav a jiných pomocných procesů, přesto však mohu tento plugin vřele doporučit. Dodělání bypassu a možnosti přimíchání do mixu jsou zajímavým zlepšením oproti minulé verzi. V příští verzi bych ocenil rozšíření nabídky efektů o grafický ekvalizér.

Zdenko Kamenický

– hudební producent, majitel produkčního studia

Testovaný plugin je velice kompaktní nástroj, který ušetří mnoho času (což je pro nás všechny tou nejcennější komoditou). Na jedné straně je uživatelsky příjemný a poradí si s ním i začátečník. Na straně druhé nabízí nepřebernou škálu možností pro profi využití. Design je adekvátní k současným trendům, neodvádí pozornost od práce. Ovšem zmíněný decentní vzhled může odrazovat fajnšmekry a příznivce všemožných blikátek, hejbátek či točítek. Přesto si troufám tvrdit, že se jedná o šikovného pomocníka, kterého budu při práci často zaměstnávat.

Příloha B

Obsah přiloženého DVD

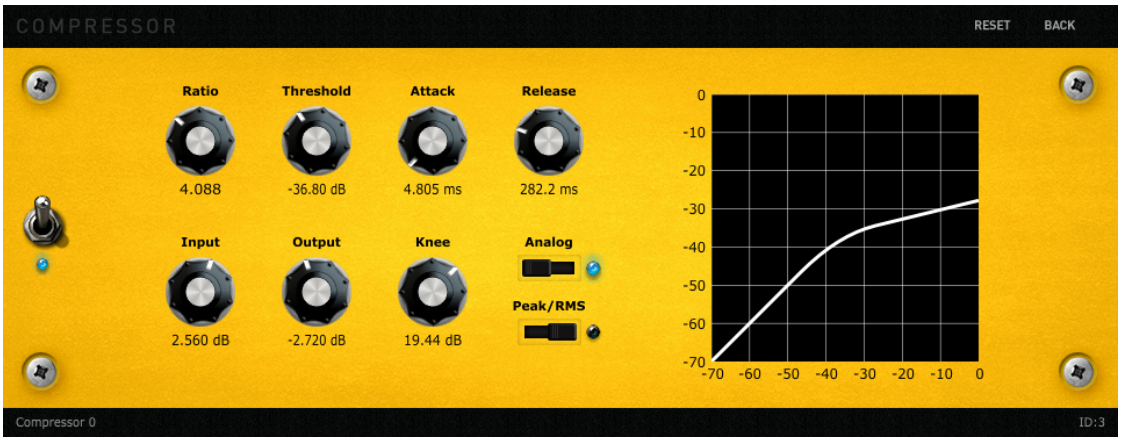
Přiložené DVD obsahuje následující soubory a adresáře:

- `pdf` – zde je uložen text technické zprávy ve formátu *pdf*,
- `latex` – adresář se zdrojovými soubory technické zprávy ve formátu LaTeX,
- `samples` – složka se zvukovými signály použitými při testování pluginu,
  - `instruments` – adresář obsahuje stopy hudebních nástrojů,
  - `vocals` – složka s vokálními stopami,
  - `signals` – složka s testovacími signály,
- `src` – adresář se zdrojovými soubory aplikace,
- `lib` – složka se zdrojovými kódy použitých knihoven,
- `build` – obsahuje výslednou aplikaci ve formátech *dll* a *vst3*,
- `using` – složka věnovaná použití pluginu,
  - `examples` – zde jsou uloženy zvukové ukázky získané použitím pluginu,
  - `video` – obsahuje video demonstrující použití pluginu,
  - `presets` – složka se soubory obsahujícími přednastavení pluginu.

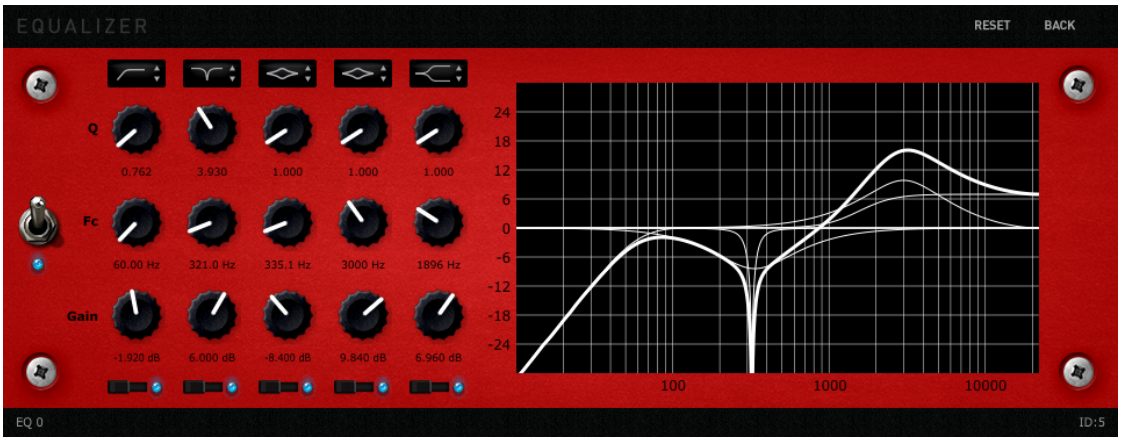
Příloha C

Obrazovky modulů

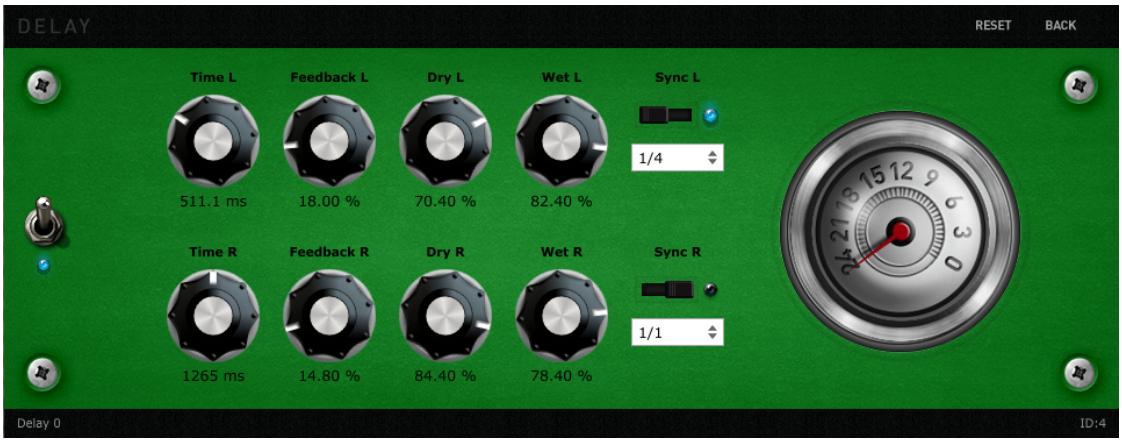
V této příloze jsou zobrazeny ovládací obrazovky vybraných modulů.



Obrázek C.1: Ovládací obrazovka kompresoru.

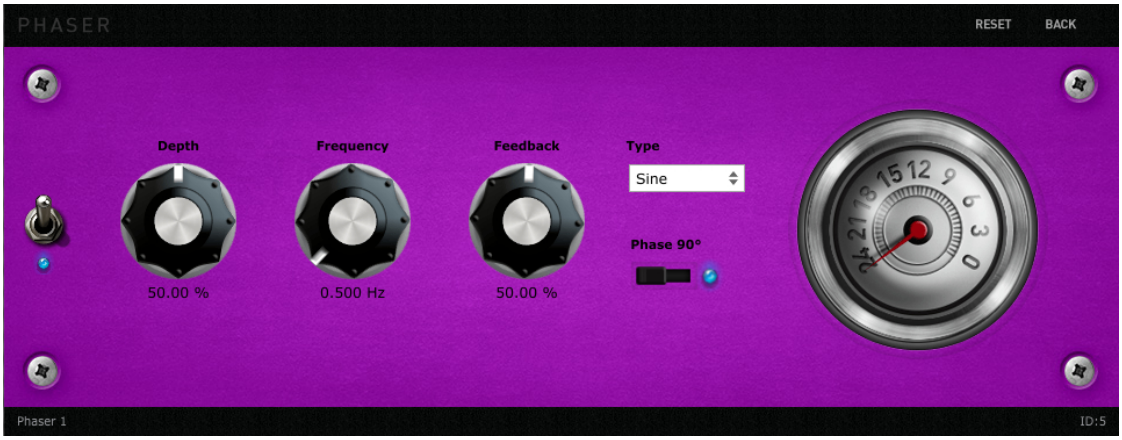


Obrázek C.2: Ovládací obrazovka ekvalizéru.



Obrázek C.3: Ovládací obrazovka efektu delay.

Obrázek C.4: Ovládací obrazovka efektu reverb.

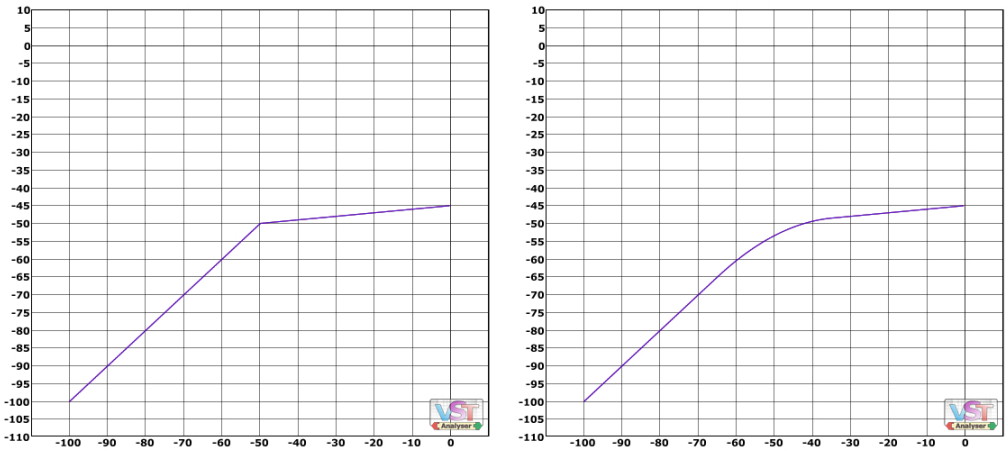


Obrázek C.5: Ovládací obrazovka efektu phaser.

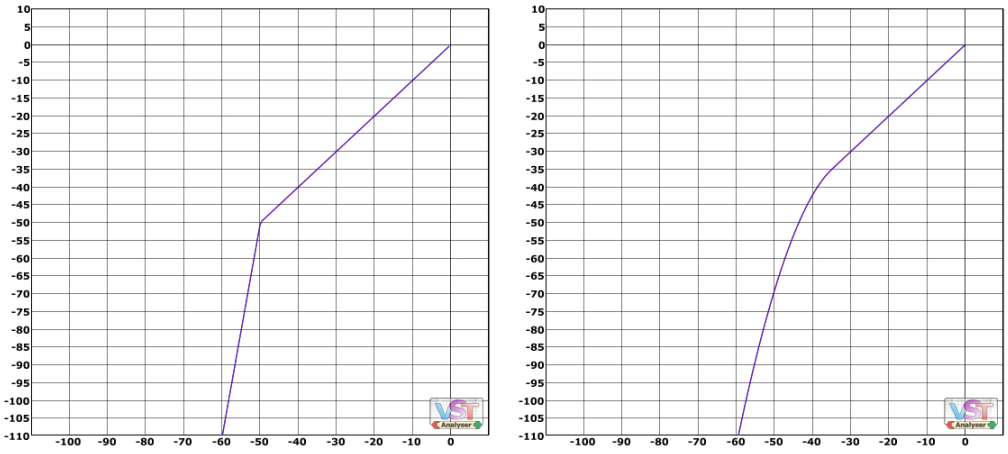
Příloha D

Měření

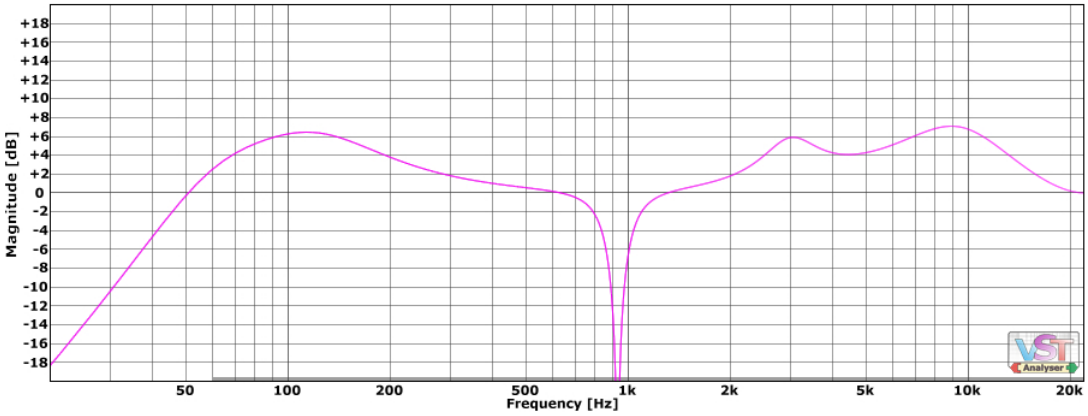
Tato příloha obsahuje dynamické a frekvenční charakteristicky vybraných modulů, jenž byly naměřeny během testování pluginu *Ptolemaios* pomocí aplikace *VST Plugin Analyser*.



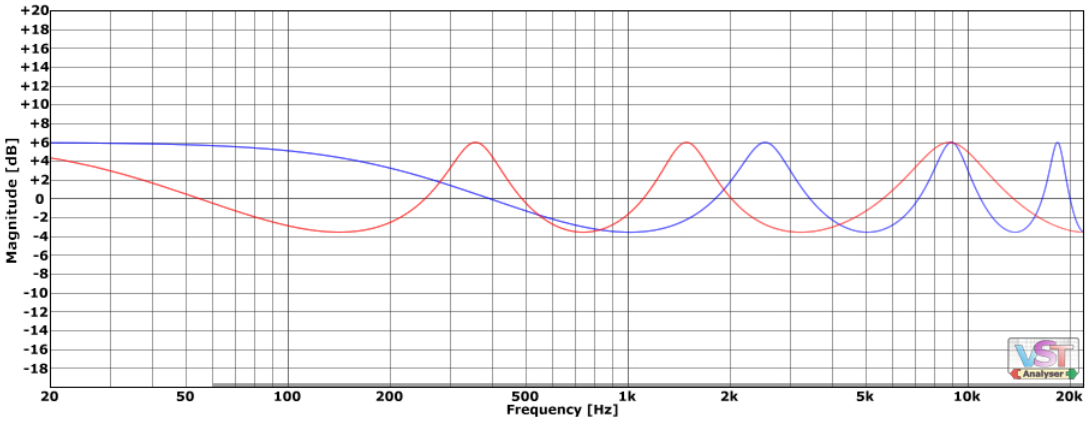
Obrázek D.1: Křivka *kompresoru* s různým nastavením šířky kolene.



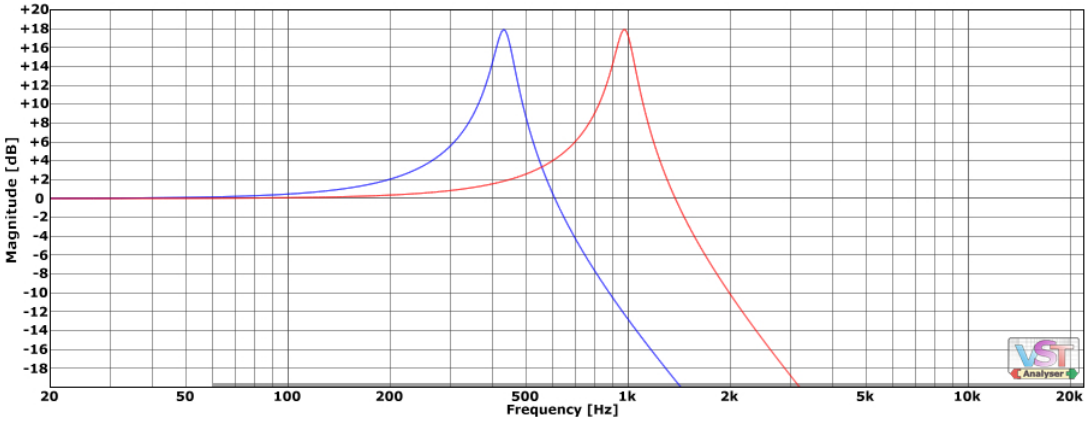
Obrázek D.2: Křivka *expanderu* s různým nastavením šířky kolene.



Obrázek D.3: Frekvenční charakteristika *ekvalizéru*.



Obrázek D.4: Zachycení frekvenční charakteristika efektu *phaser*.



Obrázek D.5: Zachycení frekvenční charakteristika efektu *wah wah*.