Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Návrh číslicového zvukového efektu a jeho implementace ve formě VST pluginu

Plzeň 2018 Tomáš Kleisner

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 14. května 2018

Tomáš Kleisner

V práci jsou použity názvy programových produktů, firem apod., které mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Poděkování

Děkuji vedoucímu práce Ing. Kamilu Ekšteinovi, Ph.D. za konzultace a cenné rady. Dále bych rád poděkoval Lukáši Maiovi, Tomáši Václavkovi a Stanislavu Boháčovi za testování pluginu.

Abstract

The goal of this thesis is to learn about Virtual Studio Technology (VST) and to design and implement digital sound effect as VST plugin. The thesis is about basic digital audio signal processing techniques used in digital effect design, available plugin design tools and frameworks and exploration of open-source plugin implementations. The result of this work is multiband distortion VST plugin with the implementation of distortion function based on a linear prediction. This plugin can be used within host DAW applications for sound recording and processing.

Abstrakt

Cílem této práce je seznámení s technologií Virtual Studio Technology (VST) a návrh a implementace číslicového zvukového efektu jako VST pluginu. Práce se zabývá základními technikami zpracování číslicového zvukového signálu s ohledem na potřeby realizace efektu, dostupnými vývojovými nástroji a frameworky a prozkoumáním open-source implementací VST pluginů. Výsledkem této práce je zvukový efekt vícepásmového zkreslení realizovaný jako VST plugin s implementací funkce pro zkreslení zvuku založené na lineární predikci, který je použitelný v hostitelských aplikacích DAW pro nahrávání a zpracování zvuku.

Obsah

1 Úvod **7**

2 Zvukový signál **8**
2.1 Digitální signál 8
2.2 MIDI 10

3 Virtual Studio Technology **11**
3.1 Digital Audio Workstation 11
3.2 Typy VST 12
3.3 VST SDK 13

4 Zpracování digitálního zvukového signálu **15**
4.1 Diskrétní Fourierova transformace 15
4.2 Zkreslení 17
4.3 Zpoždění 20
4.4 Amplitudová modulace 20
4.5 Frekvenční modulace 21
4.6 Filtry 22
4.6.1 Filtr s konečnou impulzní odezvou 23
4.6.2 Filtr s nekonečnou impulzní odezvou 26
4.7 Vícepásmové efekty 28
4.8 Lineární predikce 28

5 Vývojové nástroje **32**
5.1 JUCE 32
5.1.1 Projekt VST pluginu v JUCE 33
5.2 WDL-OL 34
5.3 RackAFX 35
5.4 VST.NET 36
5.5 Shrnutí 36

6 Open-source pluginy **38**
6.1 mda-vst 38
6.2 ADelay 40
6.3 Audio Effects 40

7 Implementace efektů v JUCE **44**
7.1 Distortion . . . 44
7.2 Filter . . . 44
7.3 Delay . . . 46
7.4 Vibrato . . . 46
7.5 Tremolo . . . 47

8 Vícepásmové zkreslení **48**
8.1 Adaptivní zkreslení . . . 49
8.2 Typy zkreslení . . . 52
8.3 Filtry . . . 53
8.4 Nastavení efektu . . . 54
8.5 Procesor . . . 55
8.6 Editor . . . 56

9 Testování **59**
9.1 Měření zpoždění . . . 59
9.2 Testování v hostitelských aplikacích . . . 60
9.3 Uživatelské testování . . . 61

10 Závěr **62**

Literatura **63**

A Obsah přiloženého DVD **65**

B Uživatelské hodnocení **66**

1 Úvod

Virtual Studio Technology (VST) je softwarové rozhraní pro integraci zvukových efektů a syntetizérů ve formě pluginů[1] s hostitelskými aplikacemi Digital Audio Workstation (DAW) pro nahrávání a zpracování zvukového signálu. Tyto pluginy hostitelským aplikacím poskytují dodatečnou funkcionalitu a často se používají jako náhrada skutečných zvukových zařízení (efektů, analyzátorů apod.) nebo hudebních nástrojů.

Tato práce se zabývá popisem základních technik reprezentace a zpracování digitálního zvukového signálu s ohledem na potřeby realizace zvukových efektů. Dále popisuje některé dostupné open-source[2] implementace zvukových efektů jako VST pluginů a dostupné nástroje a frameworky[3] používané pro vývoj VST pluginů a aplikací pro zpracování zvuku.

Výsledkem práce bude návrh a implementace vlastního zvukového efektu jako VST pluginu s použitím vhodného vývojového nástroje a otestování jeho použitelnosti ve vybraných DAW aplikacích. Zároveň bude proveden poslechový test navrženého efektu.

[1] zásuvných modulů

[2] software s otevřeným zdrojovým kódem

[3] aplikační rámce

2 Zvukový signál

Zvuk je mechanické vlnění generované zdrojem, které se šíří v hmotném prostředí a vyvolává sluchový vjem. Uvádí se, že člověk je schopný slyšet zvuk o frekvencích přibližně od 20 Hz do 20 kHz [14].

Kromě frekvence vlnění, která udává výšku tónu, je další důležitou vlastností zvuku barva, díky které lze rozpoznat zvuk různých nástrojů nebo hlasy lidí. Je dána intenzitou vyšších harmonických tónů, které zní spolu s tónem základním. Vyšší harmonické tóny jsou tóny s frekvencí rovnou celočíselnému násobku frekvence základního tónu [14]. Při návrhu zvukových efektů se tato vlastnost používá například ke zkreslení, kde dochází k přidávání vyšších harmonických k základním tónům, a vzniká tak barevně výraznější zvuk.

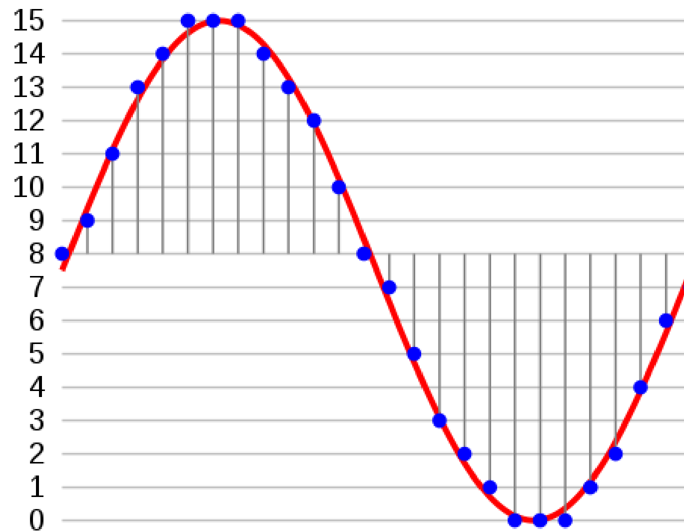
Dalším důležitým parametrem zvuku je jeho amplituda, která ovlivňuje intenzitu zvuku. K jejímu vyjádření se používá bezrozměrná logaritmická jednotka decibel (dB), která určuje poměr referenční hodnoty k hodnotě měřené (2.1). Výběr referenční amplitudy tak definuje rozsah hodnot. Při zpracování signálu se používá jako referenční hodnota maximální amplituda. Zároveň se při zpracování signálu normalizuje hodnota amplitudy od -1 do 1. Maximální amplituda pak dosahuje absolutní hodnoty 1. Po vyjádření podle vztahu (2.1), kde A_{dB} je amplituda v dB, A měřená amplituda a A_{ref} referenční amplituda, je maximální hodnota amplitudy 0 dB [11].

$$A_{dB} = 20 \log_{10} \left(\frac{|A|}{|A_{ref}|} \right) \tag{2.1}$$

2.1 Digitální signál

Zpracování digitálního zvukového signálu je jednou z aplikací zpracování digitálního signálu (angl. *Digital Signal Processing*, DSP). Mezi úlohy zpracování digitálního zvukového signálu patří například jeho reprezentace, transformace, vizualizace, analýza nebo syntéza.

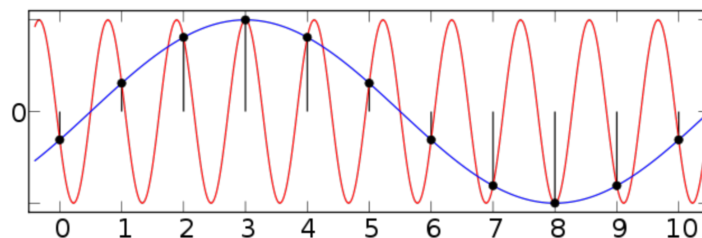
DSP systémy pracují se zvukem reprezentovaným jako pole vzorkovaných hodnot. Analogový zvukový signál je potřeba navzorkovat pomocí A/D převodníku. Standardní formát pro vzorkovaná zvuková data se nazývá Pulse Code Modulation (PCM) [11]. Metoda vzorkování spočívá v odečítání vzorků amplitudy signálu s danou vzorkovací frekvencí. Takto získané vzorky signálu jsou převedeny do binární podoby během procesu kvantizace. Parame-



Obrázek 2.1: PCM.
Zdroj: `https://commons.wikimedia.org`

trem kvantizace je bitová hloubka, která udává počet bitů pro reprezentaci jednoho vzorku. Platí, že čím vyšší jsou hodnoty vzorkovací frekvence a bitové hloubky, tím přesnější a kvalitnější je záznam. Ukázka vzorkování s bitovou hloubkou 4 bity je znázorněna na obrázku 2.1.

Minimální hodnota vzorkovací frekvence je dána Nyquistovým teorémem, který říká, že spojitý signál může být správně vzorkován jen v případě, že neobsahuje frekvence vyšší, než je polovina vzorkovací frekvence [14]. V opačném případě dochází k tzv. aliasingu a signál nemůže být správně rekonstruován při převodu zpět na analogový, protože tyto vyšší frekvence jsou špatně interpretovány. Příklad aliasingu je na obrázku 2.2, kde červená křivka znázorňuje původní signál a modrá křivka rekonstruovaný.



Obrázek 2.2: Aliasing.
Zdroj: `https://commons.wikimedia.org`

2.2 MIDI

Musical Instrument Digital Interface (MIDI) je protokol pro komunikaci elektronických hudebních nástrojů a zařízení pro digitální zpracování zvuku. V souvislosti s efektovými pluginy se MIDI používá u virtuálních nástrojů pro digitální syntézu zvuku. Na rozdíl od předchozí reprezentace zvuku jako pole vzorků se přenáší zprávy, které nesou informaci o generovaném tónu (výška, hlasitost apod.). Vzhledem k tomu, že práce se zabývá zpracováním digitálního zvukového signálu, protokol MIDI a další související techniky syntézy zvuku se v ní nepoužívají.

3 Virtual Studio Technology

Technologie VST byla vyvinuta firmou Steinberg[1] v roce 1996. Jedná se o volně dostupný vývojový kit (SDK[2]) pro vývoj zvukových pluginů, které poskytují dodatečnou funkcionalitu hostitelským programům. Díky dostupnosti pro všechny vývojáře se VST stalo nejrozšířenějším standardem pro vývoj zvukových pluginů. Nejnovějším standardem je VST3, který byl vydán 17. 1. 2008 [15]. Vedle VST se používají i další formáty a standardy jako je například Real Time AudioSuite (RTAS), Avid Audio eXtension (AAX) nebo Audio Unit (AU).

Plugin je z pohledu hostitelské aplikace černá skříňka s libovolným počtem vstupů a výstupů ve formě proudů navzorkovaného signálu nebo MIDI událostí. Proudy jsou rozděleny hostitelem do sekvence bloků, které jsou postupně předávány pluginu ke zpracování. Ačkoliv je kód VST pluginu nezávislý na platformě, podoba přeloženého pluginu závisí na architektuře a operačním systému. Pluginy se překládají a distribuují jako dynamicky linkované knihovny. Pro operační systémy Windows jsou to soubory DLL[3], pro Linux soubory SO[4] a pro Mac OS X soubory Mach-O. Hostitelským programům s podporou VST stačí znát cestu k uloženým pluginům, aby mohly být zavedeny. Některé při instalaci vytvoří adresář pro uložení pluginů, některé umožňují zadat cestu k libovolnému adresáři s pluginy.

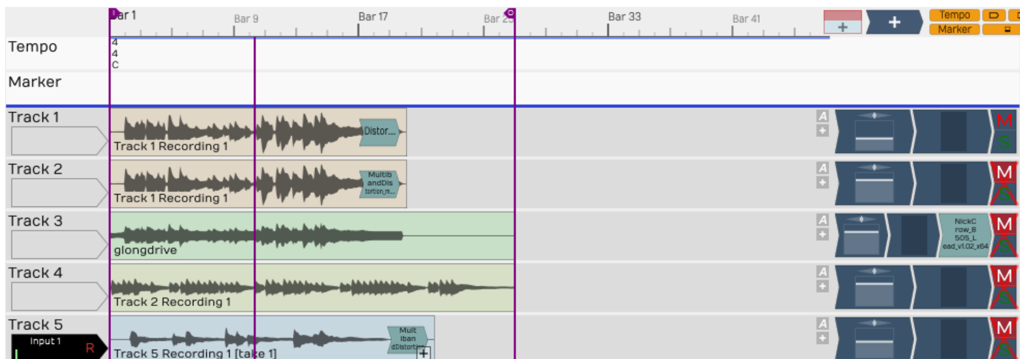
3.1 Digital Audio Workstation

Hostitelské programy se označují jako Digital Audio Workstation (DAW) a slouží pro nahrávání a editaci vícekanálového zvuku. Často se pod pojmem DAW nemyslí jen software, ale veškeré vybavení pro práci s digitálním zvukem včetně počítače a zvukové karty. Mezi známé DAW programy patří například FL Studio, Cubase, Reaper, Tracktion, Audacity, LMMS apod. Podle použití se liší dostupnost těchto programů. Kromě DAW pro profesionální použití v nahrávacích studiích jsou dostupné i programy pro domácí použití zdarma nebo jako open-source.

Uživatelské rozhraní a možnosti různých DAW programů se liší, ale často je společným znakem vícestopý editor, který připomíná vícestopý páskový

[1] `https://www.steinberg.net`
[2] Software Development Kit
[3] Dynamic-link Library
[4] Shared Object

magnetofon, a poskytuje tak hudebníkům a technikům intuitivní ovládání (příklad uživatelského rozhraní je na obrázku 3.1). Každá zvuková stopa obsahuje obvykle jeden nástroj (zpěv, kytara, bicí, ...), který může být nahraný přes zvukovou kartu nebo je ve formě MIDI. Každá stopa se dá editovat samostatně například posunem po časové ose, změnou hlasitosti, ekvalizací, stereo vyvážením nebo přidáním zvukového efektu. Úpravou a složením jednotlivých stop se vytvoří výsledné smíchání nahrávky.



Obrázek 3.1: Ukázka editoru v programu Tracktion 6

3.2 Typy VST

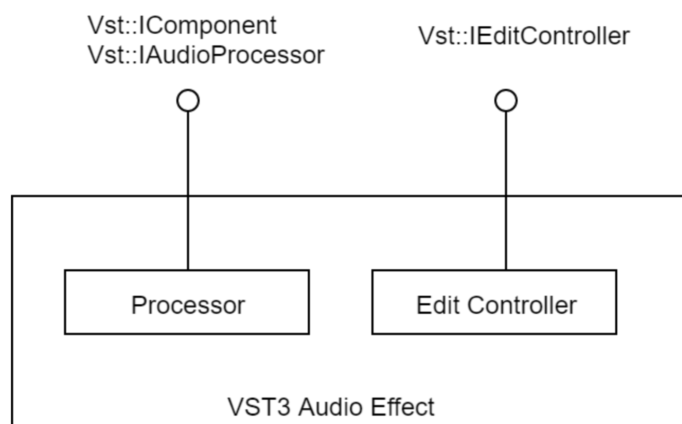
VST pluginy se zpravidla dělí podle použití na VST efekty (VSTfx) a VST nástroje (VSTi). Efekty slouží převážně k úpravě zvukového vstupu, například jako náhrada fyzických zvukových efektů jako je zkreslení, zpoždění, ozvěna apod. VST efekty mohou být pro jednu zvukovou stopu řetězeny za sebe stejně jako u fyzických zařízení, které jsou propojeny mezi sebou kabely a vytváří posloupnost mezi hudebním nástrojem a výstupním zařízením, přičemž způsob seřazení jednotlivých efektů může mít vliv na výsledný zvuk. VST pluginy nemusí nutně zvuk transformovat, ale mohou sloužit pouze k analýze a vizualizaci.

VST nástroje nahrazují reálné hudební nástroje. Jedná se například o zvukové syntetizátory a samplery, které na vstupu zpracovávají MIDI zprávy a generují zvuk nástroje. Používají se například jako náhrada za klávesy nebo bicí nástroje.

3.3 VST SDK

VST SDK je volně dostupné na webu společnosti Steingberg[5]. SDK obsahuje kompletní zdrojové kódy včetně dokumentace a vzorových projektů.

VST plugin se skládá ze dvou základních komponent pro zpracování zvuku (*Processor*) a ovládání (*Edit Controller*), které jsou znázorněny na obrázku 3.2. Prakticky jsou to dvě třídy implementující daná rozhraní. Toto rozdělení umožňuje běh obou částí v různých kontextech, dokonce i na různých počítačích [15].



Obrázek 3.2: Základní koncept VST pluginu [15]

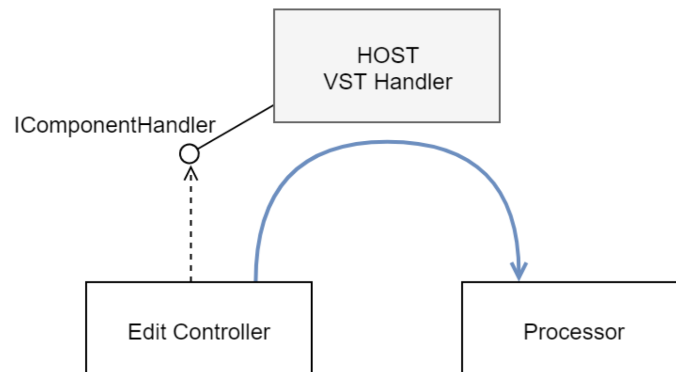
Procesor se stará o samotné zpracování signálu a implementuje dvě rozhraní z VST API. Prvním rozhraním je `Vst::IComponent`. To se stará o obecné fungování pluginu uvnitř hostitelské aplikace. Poskytuje ID jako informaci pro spojení a vytvoření odpovídajícího kontroleru. Dále poskytuje informaci o sběrnicích, které představují vstupy a výstupy pluginu.

Druhým rozhraním je `Vst::IAudioProcessor`, které reprezentuje část pluginu specifickou pro zpracování zvuku. V první řadě jde o nastavení neměnných parametrů a přechody mezi aktivním a neaktivním stavem pluginu. Podstatnou součástí je samotné zpracování signálu v metodě `process`. Zpracování běží v samostatném vlákně, a proto se všechna potřebná data předávají přes parametr metody jako způsob předcházení problémů se synchronizací [15]. Protože se signál zpracovává po blocích, předává se velikost bloku, která může být od jedné do maximální stanovené velikosti a může se v každém volání měnit. Další jsou pole se zvukovými daty, která jsou adresována indexy a přiřazena vstupním a výstupním kanálům. Tato pole jsou bloky zvukových dat poskytnutých hostitelskou aplikací ke zpracování

[5]`https://www.steinberg.net/en/company/developers.html`

a fungují jako vyrovnávací paměti. Dále se předávají parametry, kontext a události.

Kontroler implementuje `Vst::IEditController` a je zodpovědný za část pluginu spojenou s ovládáním a grafickým uživatelským rozhraním. Hostitelský program musí poskytnout rozhraní `Vst::IComponentHandler` pro komunikaci s procesorem i editorem (obr. 3.3). Při návrhu pluginu je potřeba počítat s faktem, že obě komponenty pracují v samostatných vláknech a v některých případech může při editaci dojít k nežádoucímu souběhu.



Obrázek 3.3: Komunikace mezi editorem a procesorem [15]

4 Zpracování digitálního zvukového signálu

V následující kapitole jsou popsány techniky a postupy zpracování digitálního zvukového signálu a realizace vybraných zvukových efektů.

4.1 Diskrétní Fourierova transformace

Způsob digitalizace analogového signálu pomocí PCM vede k reprezentaci signálu v časové doméně jako pole po sobě jdoucích vzorků, kde každý vzorek určuje hodnotu amplitudy v čase. Jiný způsob reprezentace signálu je ve frekvenční doméně, která zobrazuje energii frekvenčních pásem signálu.

Pro převod mezi signálem v časové a ve frekvenční doméně slouží matematická metoda zvaná Fourierova transformace. Její podoba pro digitální signál se nazývá diskrétní Fourierova transformace (DFT) a je definována vztahem (4.1) nebo s převodem pomocí Eulerova vztahu ve tvaru (4.2).

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{\frac{-j2\pi kn}{N}} \quad k = 0, ..., N-1 \tag{4.1}$$

$$X[k] = \sum_{n=0}^{N-1} x[n] (cos(\frac{2\pi kn}{N}) - jsin(\frac{2\pi kn}{N})) \quad k = 0, ..., N-1 \tag{4.2}$$

Tento vztah lze chápat jako korelaci mezi vstupním signálem x a rotací fázoru o frekvenci $\frac{k}{N}$. X je odhad spektrální hustoty vstupního signálu, jehož prvky $X[k]$ jsou komplexními čísly, které obsahují informaci o amplitudě a fázi jednoho fázoru o dané frekvenci [9]. Energie frekvenčního pásma v signálu se vyjadřuje jako magnituda. Na rozdíl od amplitudy je určena jako absolutní hodnota podle vzorce (4.3), případně v decibelech podle vzorce (4.4).

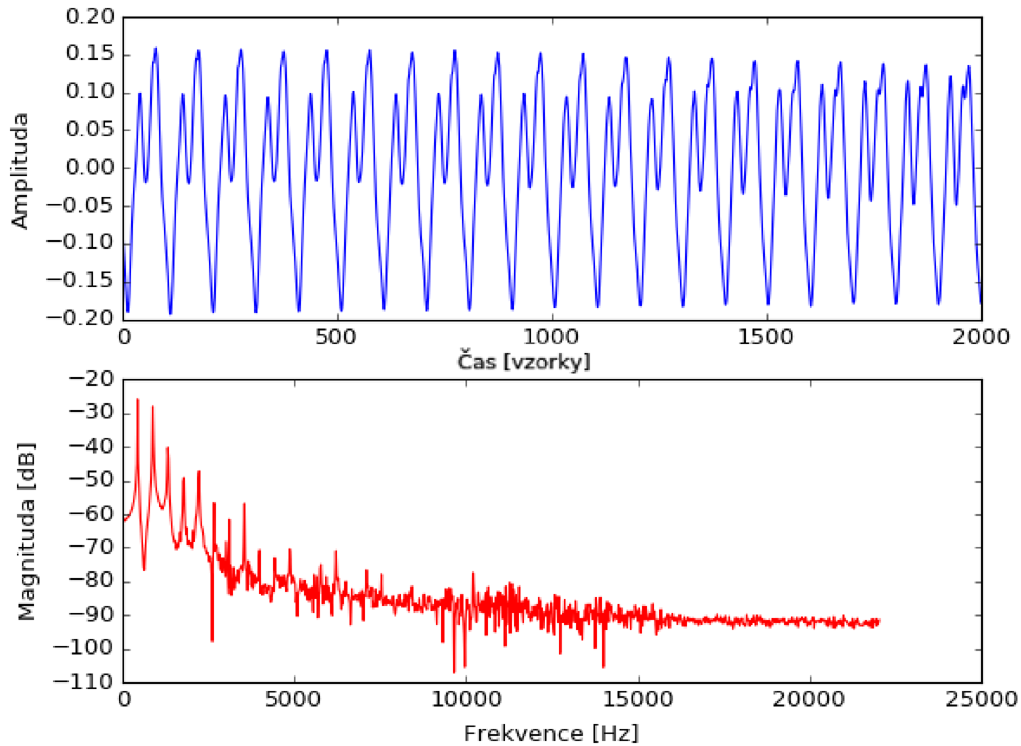
$$Mag[k] = \sqrt{{Re_{X[k]}}^2 + {Im_{X[k]}}^2} \tag{4.3}$$

$$Mag[k]_{dB} = 20 \log_{10} (Mag[k]) \tag{4.4}$$

Frekvenční pásmo $X[k]$ na indexu k v herzích se získá s pomocí vzorkovací frekvence f_s podle vzorce (4.5).

$$f[k]_{Hz} = f_s \frac{k}{N} \tag{4.5}$$

Ukázka DFT je na obrázku 4.1. Horní graf znázorňuje průběh signálu v časové doméně (komorní A - 440 Hz na flétnu) a spodní znázorňuje odhad spektrální hustoty.



Obrázek 4.1: Diskrétní Fourierova transformace

Výše popsaný postup slouží k analýze digitálního signálu. Použitím inverzní DFT (IDFT) podle vztahu (4.6) se převede signál z frekvenční domény do časové. Díky této vlastnosti lze vybrané složky frekvenčního spektra signálu potlačit nebo zesílit, a použít tak DFT a následně IDFT jako nástroj pro filtrování a ekvalizaci.

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{\frac{j2\pi kn}{N}} \tag{4.6}$$

Pro výpočet DFT se používá algoritmus Fast Fourier Transform (FFT). Jedná se o efektivní metodu výpočtu, protože složitost FFT je $O(n \log_2 n)$,

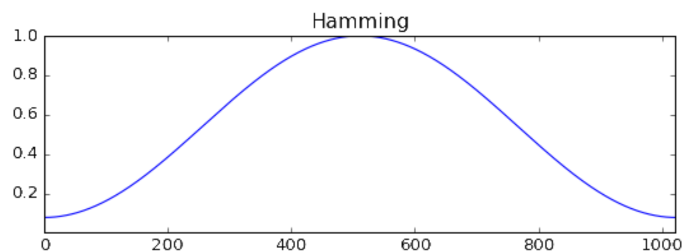
zatímco výpočet DFT má složitost $O(n^2)$ [14]. Algoritmus FFT je implementován v mnoha volně dostupných knihovnách jako je například Numpy[1] pro Python nebo FFTW[2] pro `C` a `C++`.

Frekvenční spektrum se na delších úsecích zvukového signálu může měnit. Aplikací DFT na velké časové okno nelze zaznamenat změny frekvenčního spektra v čase. Dále pak není možné DFT aplikovat při proudovém zpracování signálu ve VST pluginu, protože analyzovaný úsek musí být konečný. Pro tyto případy se používá metoda nazývaná Short-Time Fourier Transform (STFT). STFT je aplikace DFT na signál postupně po krátkých blocích stejné délky N, které se překrývají a násobí tzv. vyhlazovacím oknem w. STFT může být definována vztahem (4.7), kde l je číslo okna a H velikost posuvu, který určuje míru překrytí.

$$X_l[k] = \sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} w[n]x[n+lH]e^{\frac{-j2\pi kn}{N}} \quad l = 0, 1, 2, \ldots \tag{4.7}$$

Aplikací vyhlazovacího okna se odstraní nespojitosti na koncích analyzovaného bloku signálu potlačením amplitudy. Tím se zmírňuje jev tzv. prosakování (*leakage*) [9]. Při prosakování u DFT dochází k tomu, že se zkoumaná frekvenční složka signálu projevuje i na sousedních pozicích, než jen na daném indexu k. Používanými vyhlazovacími okny jsou například Hannovo nebo Hammingovo (obr. 4.2) podle vztahu (4.8).

$$w[n] = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right) \tag{4.8}$$



Obrázek 4.2: Hammingovo okno

4.2 Zkreslení

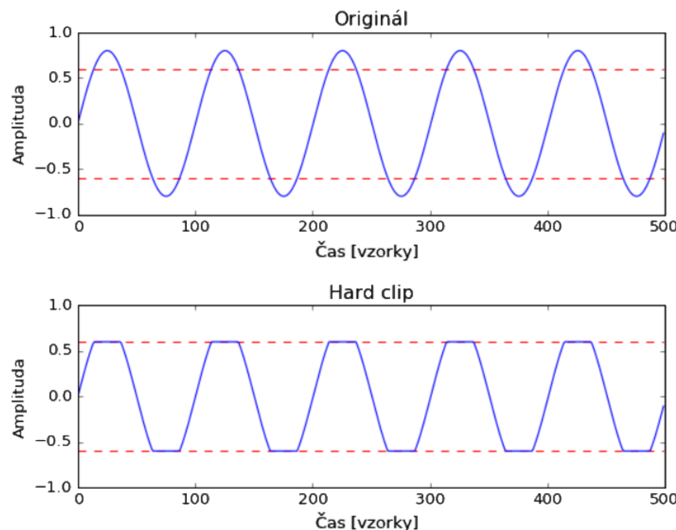
Efekt zkreslení (*distortion*) má široké využití při hře na elektrickou kytaru. Kytarové zkreslení se objevuje v mírnější podobě například v blues, silné

[1]`http://www.numpy.org`
[2]`http://www.fftw.org`

zkreslení je patrné u kytar například v rocku, punku nebo metalu. Původně se kytarové zkreslení objevovalo jako důsledek zesílení signálu u prvních kytarových zesilovačů a reproduktorů. V dnešní době se vyrábí zesilovače uzpůsobené pro vytvoření specifického zkresleného tónu nebo pedálové efekty, které zkreslují signál ještě před vstupem do zesilovače.

Zkreslení je založeno na použití nelineární funkce, která deformuje tvar vrcholů vlny vstupního signálu. Tím dochází k přidání množství vyšších harmonických frekvencí k vstupnímu signálu a vzniká tak barevnější a zkreslený zvuk. U kytarových efektů jsou základními typy zkreslení *distortion*, *overdrive* a *fuzz*. Ty jsou často zaměňovány, ale mají odlišný způsob zkreslení a produkují jiný zvuk.

Nejjednodušším způsobem změny tvaru vlny, který používá efekt *distortion*, je tzv. *hard clipping*. Je definována funkce (4.9), jejíž parametry jsou vzorek vstupního signálu x a prahová hodnota t (*threshold*), která určuje hranici maximální hodnoty amplitudy. U signálu nad prahovou hodnotou dochází k ořezávání, které je znázorněno na obrázku 4.3, kde v horním grafu je původní čistý signál a na spodním zkreslený. Červená čára znázorňuje prahovou hodnotu. Ta může mít hodnotu rovnou jedné $t = 1$, tedy maximální normalizované amplitudy. Ke zkreslení pak dochází zesílením signálu před jeho zpracováním.

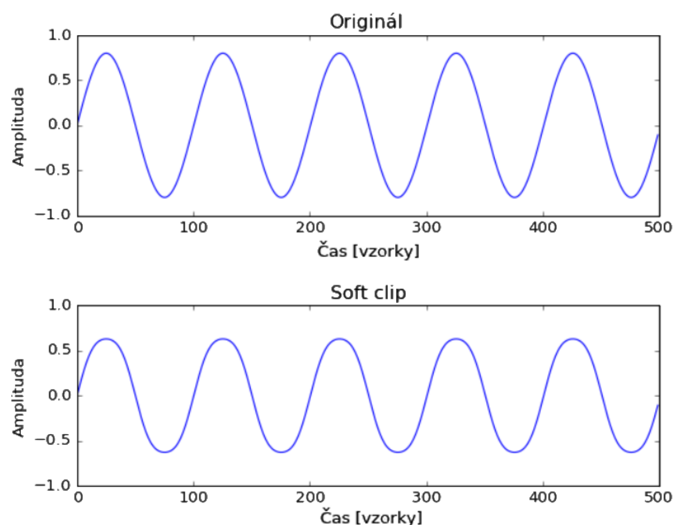


Obrázek 4.3: Hard clipping

$$f(x,t) = \begin{cases} -t, & x \leq -t \\ x, & -t \leq x \leq t \\ t, & x \geq t \end{cases} \tag{4.9}$$

Efekty typu *fuzz* jsou efekty s největším zkreslením způsobeným velkým zesílením a funkcí, která transformuje výstup podobný obdélníkové vlně. Zvuk efektu může v některých případech připomínat zvuk poškozeného reproduktoru.

Jiný způsob změny tvaru vlny, používaný u efektů *overdrive*, je tzv. *soft clipping*. Na rozdíl od předchozího produkuje o něco jemnější zkreslení, protože oříznutí hran má hladší průběh, jak je znázorněno na obrázku 4.4. Využívá se například kubické nelinearity [12] podle funkce (4.10), nebo exponenciální funkce jako například (4.11).



Obrázek 4.4: Soft clipping

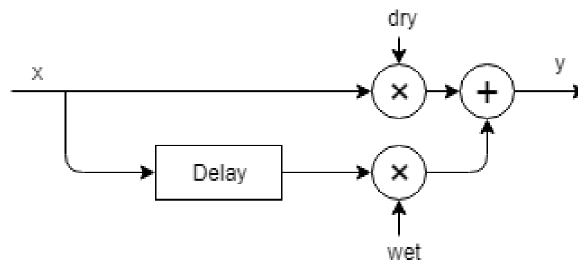
$$f(x) = \begin{cases} -\frac{2}{3}, & x \leq -1 \\ x - \frac{x^3}{3}, & -1 \leq x \leq 1 \\ \frac{2}{3}, & x \geq 1 \end{cases} \tag{4.10}$$

$$f(x) = \begin{cases} 1 - e^{-x}, & x > 0 \\ -1 + e^{x}, & x \leq 0 \end{cases} \tag{4.11}$$

Reálné efektové pedály pro zkreslení jsou vyráběny v mnoha variantách různými výrobci. Jejich ovládací prvky pro nastavení parametrů se mohou lišit. Často se na nich objevuje nastavení zkreslení (*drive*), které ovlivňuje zesílení a prahovou hodnotu, nastavení tónu (*tone*) a úroveň výstupního signálu (*level*).

4.3 Zpoždění

Zpoždění (*delay*) je další z řady efektů používaných při zpracování zvuku. Tato technika se používá k tvorbě dalších efektů nebo samostatně, kdy může navozovat dojem ozvěny. Základní myšlenkou je vstupní signál rozdělit na dva proudy, kde jeden jde přímo na výstup a druhý se zpozdí o určenou dobu. Na výstupu se oba proudy sečtou v daném poměru. V oblasti tvorby zvukových efektů se tento poměr označuje jako *wet/dry mix*, kde *wet* je množství upraveného a *dry* množství čistého signálu. Základní schéma efektu zpoždění je na obrázku 4.5.



Obrázek 4.5: Zpoždění - delay

Zpoždění může být při zpracování digitálního zvukového signálu implementováno pomocí cyklické vyrovnávací paměti, kam se ukládají vzorky vstupního signálu. Funkci pro dosažení efektu zpoždění lze zjednodušeně napsat podle vztahu (4.12), kde y je výstup, x vstup, f_s vzorkovací frekvence v Hz, t_d čas zpoždění v sekundách, n index vzorku a m je hodnota *dry* mixu v rozsahu od 0 do 1.

$$y[n] = mx[n] + (1 - m)x[n - f_s t_d] \tag{4.12}$$

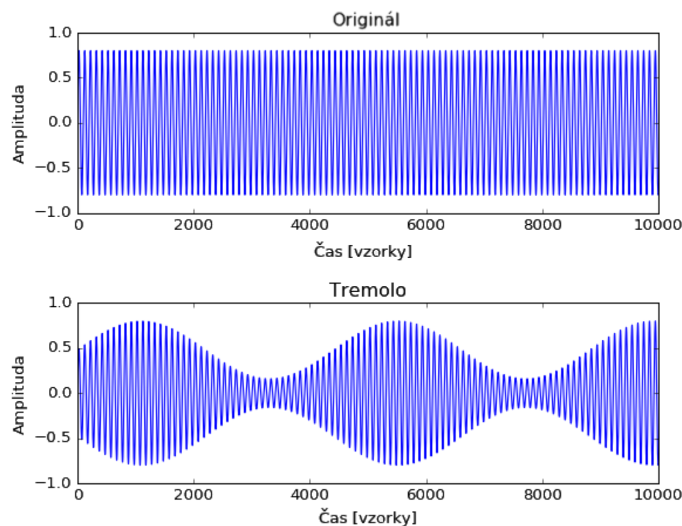
4.4 Amplitudová modulace

Efekt, při kterém se cyklicky mění amplituda, se nazývá *tremolo*. K vytvoření efektu se používá zařízení zvané Low Frequency Oscillator (LFO), které produkuje pomalu oscilující sinusoidu obvykle o frekvencích do 20 Hz [16]. K dosažení efektu při zpracování digitálního signálu stačí vynásobit vstupní signál signálem generovaným LFO podle vztahu 4.13, kde y je výstupní signál, x vstupní signál, A_{min} je minimální násobek amplitudy, A_{max} maximální násobek amplitudy, f frekvence LFO, n index vzorku a f_s vzorkovací frekvence. Platí, že čas v sekundách je $t = \frac{n}{f_s}$ a součet amplitud $A_{min} + A_{max} = 1$.

LFO je posunuto tak, aby se generovaly hodnoty od 0 do 1.

$$y[n] = x[n](A_{min} + (A_{max} - A_{min})(0.5 + 0.5\sin(2\pi ft))) \tag{4.13}$$

Výsledek efektu je viditelný při vizualizaci v časové doméně. Obrázek 4.6 znázorňuje výstup efektu s parametry $f = 10$ Hz, $A_{min} = 0.2$ a $A_{max} = 1$ při vzorkovací frekvenci 44100 Hz.



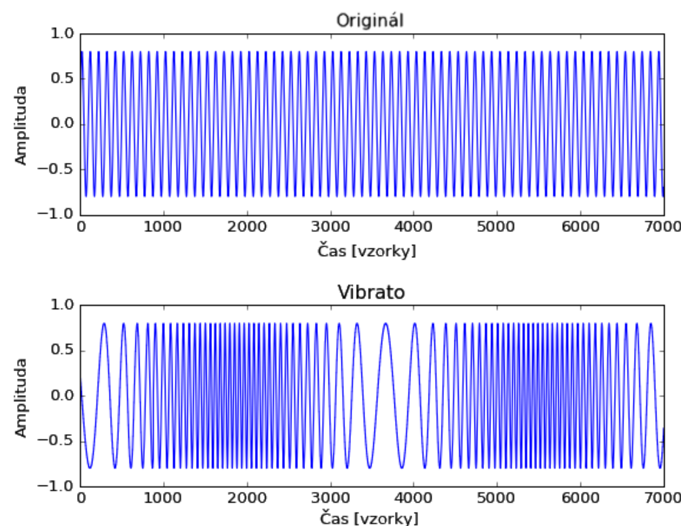
Obrázek 4.6: Tremolo

4.5 Frekvenční modulace

Podobně jako u předchozího efektu se dá LFO využít pro periodickou změnu frekvence. Takový efekt se nazývá *vibrato* a přirozeně vzniká například při hře na elektrickou kytaru, kdy se po úderu na strunu a rozeznění tónu opakovaně prstem na hmatníku vytahuje struna a vrací zpět do původní polohy.

K dosažení efektu se používá kombinace LFO a efektu zpoždění. Parametrem je maximální zpoždění t_{max} a frekvence LFO f. Funkce pro efekt se dá popsat podle vztahu (4.14). Periodická změna frekvence je viditelná v časové doméně. Výsledek efektu je znázorněn na obrázku 4.7, kde je frekvence LFO $f = 12$ Hz a maximální zpoždění $t_{max} = 0,02$ s.

$$y[n] = x[n - f_s t_{max}(0.5 + 0.5\sin(2\pi ft))] \tag{4.14}$$



Obrázek 4.7: Vibrato

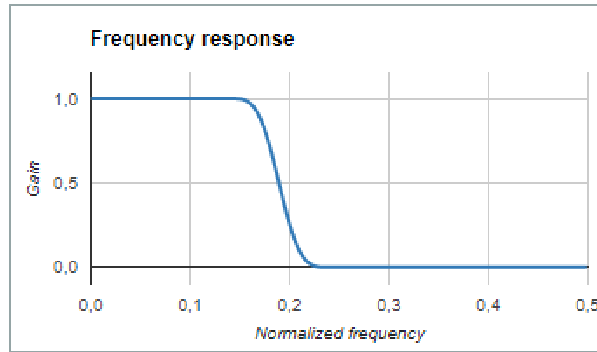
4.6 Filtry

Filtr je lineární a časově invariantní systém pro úpravu obsahu frekvenčních pásem signálu. Pro lineární systém platí, že pokud $x_1(n) \to y_1(n)$ a $x_2(n) \to y_2(n)$, pak $ax_1(n) + bx_2(n) \to ay_1(n) + by_2(n)$, kde $a, b \in \mathbb{R}$ [4]. Časově invariantní znamená, že se chování filtru nemění v čase a vždy závisí na vstupu nebo jeho nastavení.

Digitální filtry se používají například pro odstranění šumu, aliasingu nebo selekci frekvenčního spektra signálu, kde mají za úkol potlačovat nebo propouštět určitá frekvenční pásma. Filtry, které propouštějí frekvenční pásma, se dělí na dolní propust (*low-pass*), horní propust (*high-pass*) a pásmovou propust (*band-pass*).

Dolní propust je filtr s takovou frekvenční odezvou, která nepropouští frekvence nad zlomovou frekvencí f_c. Frekvenční odezva filtru (obr. 4.8) je definována jako spektrum výstupního signálu dělené spektrem vstupního signálu [10]. Obdobně horní propust propouští jen frekvence nad zlomovou frekvencí a pásmová propouští frekvence v určeném frekvenčním pásmu.

Frekvenční odezva H charakterizuje filtr ve frekvenční doméně. Stejně tak může být filtr charakterizován impulzní odezvou h v časové doméně. Impulzní odezva je výstupní signál filtru při zpracování jednotkového impulzu δ. Ten je definován podle vztahu (4.15) jako signál s jedním vzorkem o amplitudě rovné jedné v čase nula, který je následován vzorky s nulovou amplitudou [10]. Tento vztah se používá jako diskrétní podoba Diracova



Obrázek 4.8: Frekvenční odezva filtru
Zdroj: `https://fiiir.com`

impulsu, který je pro spojitý signál definován vztahem (4.16) [8].

$$\delta(n) = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases} \tag{4.15}$$

$$\delta(x) = \begin{cases} \infty, & x = 0 \\ 0, & x \neq 0 \end{cases}, \quad \int_{-\infty}^{\infty} \delta(x)dx = 1 \tag{4.16}$$

Protože obě charakteristiky nesou úplnou informaci o filtru, jsou mezi sebou převoditelné pomocí Fourierovy transformace a její inverze. Frekvenční odezva je Fourierovou transformací impulzní odezvy [14]. Protože je konvoluce signálu s impulzní odezvou v časové doméně ekvivalentní násobení frekvenčního spektra signálu s frekvenční odezvou, platí vztah (4.17) pro popis filtru ve frekvenční doméně a vztah (4.18) pro popis filtru v časové doméně [14].

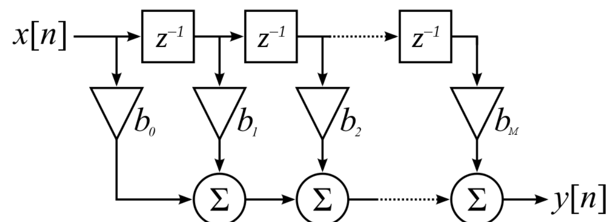
$$y(n) = x(n) * h(n) \tag{4.17}$$

$$Y(f) = X(f) \times H(f) \tag{4.18}$$

4.6.1 Filtr s konečnou impulzní odezvou

Jak bylo popsáno v předchozím textu, filtrování signálu lze realizovat pomocí DFT a inverzní DFT aplikované postupně po malých blocích signálu. Tento přístup filtruje signál ve frekvenční doméně. Dále se k filtrování signálu v časové doméně používají filtry Finite Impulse Response (FIR) nebo Infinite Impulse Response (IIR).

Na obrázku 4.9 je schéma FIR filtru o délce $N = M+1$. Symbol z^{-n} značí zpoždění jednoho vzorku. Zpoždění o jeden vzorek je $z^{-1}x(n) = x(n-1)$. Symboly b_n značí koeficienty filtru a impulzní odezva FIR filtru je definována vztahem (4.19) [10].



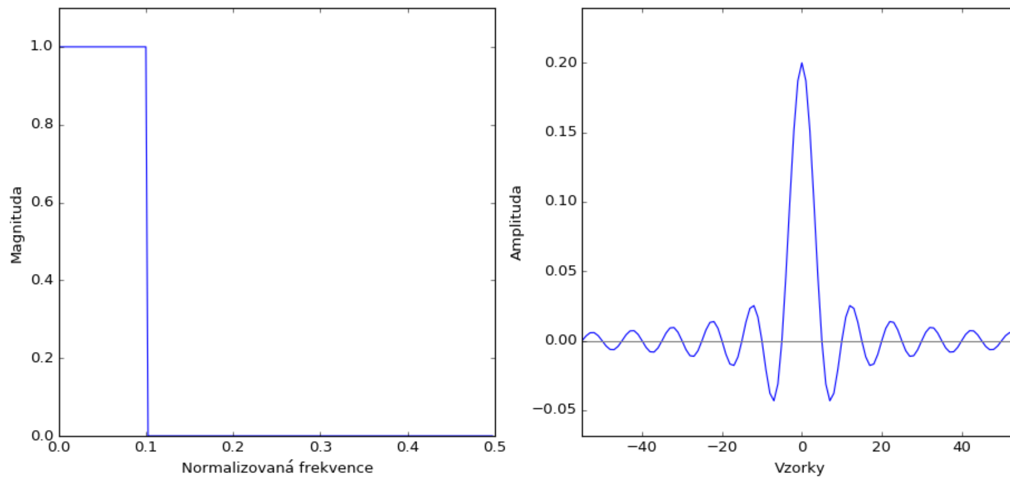
Obrázek 4.9: FIR filtr
Zdroj: `https://commons.wikimedia.org`

$$h(n) = \begin{cases} 0, & n < 0 \\ b_n, & 0 \leq n \leq M \\ 0, & n > M \end{cases} \tag{4.19}$$

FIR filtr pracuje na principu konvoluce impulzní odezvy h se vstupním signálem x a produkuje filtrovaný signál y. Impulzní odezvu si proto lze představit jako masku aplikovanou na vstupní signál. Pro výstupní signál platí vztah (4.20) [10].

$$\begin{aligned} y(n) &= b_0x(n) + b_1x(n-1) + \ldots + b_Mx(n-M) \\ &= \sum_{m=0}^{M} b_m x(n-m) \\ &= \sum_{m=-\infty}^{\infty} h(m)x(n-m) \end{aligned} \tag{4.20}$$

Pro realizaci konkrétního typu filtru záleží na volbě koeficientů b_n. Při návrhu frekvenčně selektivních filtrů (např. *low-pass*) se využívá funkce *sinc*. Tato metoda vychází z toho, že frekvenční odezva ideálního *low-pass* filtru má podobu obdélníkové funkce, kde hodnoty pod zlomovou frekvencí f_c jsou rovny jedné, hodnoty nad ní rovny nule a přechodové pásmo kolem zlomové frekvence je nekonečné malé. Na obrázku 4.10 je vlevo znázorněna frekvenční odezva ideálního *low-pass* filtru, kde je frekvence v normalizovaném tvaru jako podíl vzorkovací frekvence. Inverzní Fourierova transformace frekvenční odezvy ve tvaru obdélníkové funkce má tvar funkce *sinc* podle vztahu (4.21) [7, 13], která je znázorněná na obrázku 4.10 vpravo.

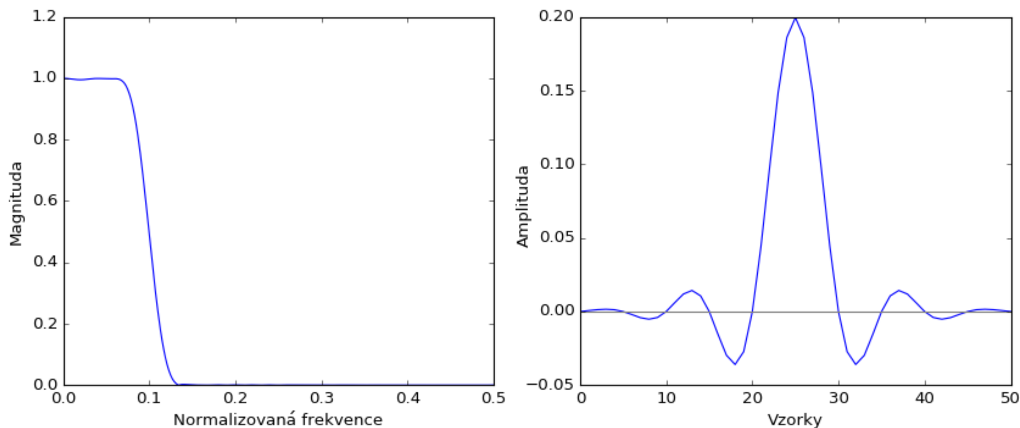


Obrázek 4.10: Frekvenční a impulzní odezva ideálního *low-pass* filtru

$$h[n] = \frac{\sin(2\pi f_c n)}{n\pi} = 2f_c \operatorname{sinc}(2f_c n) \tag{4.21}$$

Funkce *sinc* osciluje kolem nuly do nekonečna. Proto se volí konečná délku filtru a aplikuje funkce vyhlazovacího okna. Na volbě délky filtru závisí přesnost filtru a šířka přechodového pásma kolem zlomové frekvence. Aproximaci délky filtru N pro požadovanou šířku přechodového pásma lze získat ze vztahu (4.22), kde b je podíl šířky přechodového pásma a vzorkovací frekvence [7].

$$b \approx \frac{4}{N} \tag{4.22}$$



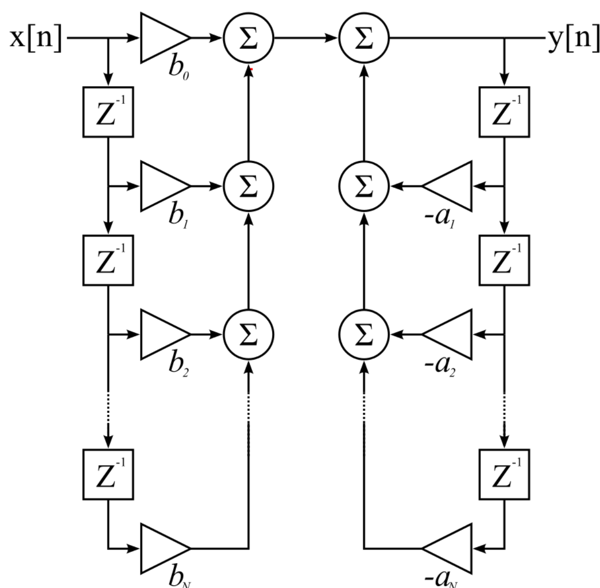
Obrázek 4.11: Frekvenční a impulzní odezva *low-pass* filtru

Na obrázku 4.11 je vlevo znázorněná frekvenční odezva *low-pass* filtru s normalizovanou zlomovou frekvencí $f_c = 0,1$ a normalizovanou přecho-

dovou šířkou pásma $b = 0,08$. Vpravo je impulzní odezva filtru vyhlazená Hammingovým oknem.

4.6.2 Filtr s nekonečnou impulzní odezvou

Na rozdíl od konečné impulzní odezvy FIR filtru, která dosáhne nulové hodnoty v čase daném počtem koeficientů, je impulzní odezva IIR filtru nekonečná, protože IIR filtr obsahuje zpětnou vazbu. IIR filtry mohou mít různou strukturu, z nichž jedna je znázorněna na obrázku 4.12.



Obrázek 4.12: IIR filtr
Zdroj: `https://commons.wikimedia.org`

IIR filtr je popsán obecně diferenční rovnicí (4.23) pro výpočet výstupního vzorku na pozici n na základě předchozích a aktuálního vstupního vzorku a předchozích výstupních vzorků [10].

$$y[n] = \sum_{i=0}^{M} b_i x[n-i] - \sum_{j=1}^{N} a_j y[n-j] \tag{4.23}$$

Pro návrh koeficientů filtru existuje několik metod. Návrh je založen na matematické metodě zvané z-transformace [4, 10] a nalezení přenosové funkce $H(z)$, pomocí které lze realizovat převod mezi koeficienty a frekvenční odezvou filtru nebo napodobit analogový filtr [14]. Pro diskrétní signál je z-

transformace dána vztahem (4.24), kde z je komplexní proměnná [4].

$$X(z) = \sum_{n=-\infty}^{\infty} x(n)z^{-n} \tag{4.24}$$

Přenosová funkce je definována jako podíl z-transformace výstupního signálu $Y(z)$ a z-transformace vstupního signálu $X(z)$ (4.25) [10]. Frekvenční odezva se získá vyčíslením polohy bodů na jednotkové kružnici podle vztahu (4.26) [4].

$$H(z) = \frac{Y(z)}{X(z)} \tag{4.25}$$

$$H(f) = H(z)\big|_{z=e^{j2\pi f}} \tag{4.26}$$

Diferenční rovnice zapsaná s pomocí z-transformace má tvar (4.27). Úpravami podle (4.28) s označením a dosazením polynomů $A(z)$ a $B(z)$ podle (4.29) vznikne tvar rovnice, ze kterého lze vyjádřit přenosovou funkci pouze pomocí koeficientů diferenční rovnice (4.30) jako podíl $B(z)$ a $A(z)$.

$$Y(z) = \sum_{i=0}^{M} b_i X(z)z^{-i} - \sum_{j=1}^{N} a_j Y(z)z^{-j} \tag{4.27}$$

$$\begin{aligned} Y(z) + \sum_{j=1}^{N} a_j Y(z)z^{-j} &= \sum_{i=0}^{M} b_i X(z)z^{-i} \\ Y(z)(1 + \sum_{j=1}^{N} a_j z^{-j}) &= X(z)\sum_{i=0}^{M} b_i z^{-i} \end{aligned} \tag{4.28}$$

$$\begin{aligned} A(z) &= 1 + \sum_{j=1}^{N} a_j z^{-j} \\ B(z) &= \sum_{i=0}^{M} b_i z^{-i} \end{aligned} \tag{4.29}$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{B(z)}{A(z)} \tag{4.30}$$

Přenosovou funkci lze také zapsat ve tvaru (4.31), kde se kořeny čitatele q_i nazývají nuly a kořeny jmenovatele p_j póly [10]. Filtr je stabilní, pokud všechny póly leží uvnitř jednotkové kružnice [4].

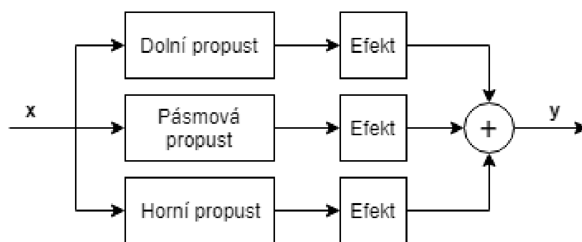
$$H(z) = b_0 \frac{(1 - q_1 z^{-1})(1 - q_2 z^{-1}) \ldots (1 - q_M z^{-1})}{(1 - p_1 z^{-1})(1 - p_2 z^{-1}) \ldots (1 - p_N z^{-1})} \tag{4.31}$$

V porovnání s FIR jsou IIR filtry výkonnější, protože lze dosáhnout stejných vlastností IIR filtru s nižším řádem než FIR filtr, ale jsou složitější na

návrh. Na druhou stranu není zaručena stabilita, což se může projevit tak, že hlasitost výstupu se postupně zvětšuje kvůli zpětné vazbě a špatnému návrhu koeficientů.

4.7 Vícepásmové efekty

Rozdělením vstupního signálu do více frekvenčních pásem a jejich samostatné zpracování může přinést nové možnosti zpracování zvuku. Příklad možného efektu je na obrázku 4.13. Signál je rozdělen na tři pásma přes dolní, pásmovou a horní propust. Na každé pásmo se dají aplikovat různé efekty s různými parametry a výsledek se nakonec sloučí zpět dohromady. Počet frekvenčních pásem, uspořádání filtrů a hranice mezi pásmy se mohou lišit.



Obrázek 4.13: Vícepásmový efekt

Tento přístup se používá například u efektu zkreslení, kde se každé pásmo zpracuje s různou mírou zkreslení nebo různou funkcí [3]. To přináší více způsobů nastavení zkreslení výsledného zvuku. Další možností je aplikovat na pásma různá nastavení zpoždění [1] nebo hlasitosti a zesílit tak daná frekvenční pásma oproti ostatním.

4.8 Lineární predikce

Lineární predikce je způsob odhadu následujícího vzorku signálu z několika předchozích vzorků. Algoritmus lineární predikce Linear Predictive Coding (LPC) se obvykle používá například k analýze a kompresi zvukového signálu při přenosu hlasu. Vychází z faktu, že se odhadovaný prvek $\hat{x}(n)$ dá vyjádřit jako lineární kombinace předchozích známých prvků $x(n-i)$ podle vztahu (4.32) [4]. Před odhadem se známé prvky analyzují a hledají koeficienty a_i pro modelování signálu.

$$\hat{x}(n) = -\sum_{i=1}^{k} a_i x(n-i) \tag{4.32}$$

Chyba predikce $e(n)$ je dána jako rozdíl mezi skutečnou a předpovězenou hodnotou [4] a je vyjádřena vztahem (4.33). Při hledání koeficientů se používá metoda nejmenších čtverců jako minimalizace součtu druhých mocnin chyby (vztah (4.34)) [2]. E značí tzv. energii chyby.

$$e(n) = x(n) - \hat{x}(n) = x(n) + \sum_{i=1}^{k} a_i x(n-i) \tag{4.33}$$

$$E = \sum_{n=-\infty}^{\infty} e^2(n) \tag{4.34}$$

Pro nalezení minima je potřeba parciálně derivovat výraz E pro každé a a položit rovno nule. Postup nalezení minima je v rovnici (4.35) pro $j = 1, \ldots, k$, kde je použito zjednodušení výrazu pro energii chyby s definováním koeficientu $a_0 = 1$. Přestože je suma zapsána jako nekonečná, v určitém bodě jsou všechny členy nulové, a proto lze přepsat rovnici do tvaru (4.36) [2].

$$\begin{aligned} \frac{\delta E}{\delta a_j} &= \frac{\delta \sum_{n=-\infty}^{\infty} \left(\sum_{i=0}^{k} a_i x(n-i) \right)^2}{\delta a_j} \\ &= \sum_{n=-\infty}^{\infty} \frac{\delta \left(\sum_{i=0}^{k} a_i x(n-i) \right)^2}{\delta a_j} \\ &= \sum_{n=-\infty}^{\infty} 2x(n-j) \sum_{i=0}^{k} a_i x(n-i) \\ &= 0 \end{aligned} \tag{4.35}$$

$$\sum_{i=0}^{k} a_i \sum_{n=-\infty}^{\infty} x(n)x(n+j+i) = 0 \tag{4.36}$$

Definováním $R(l)$ podle (4.37) a dosazením do vztahu (4.36) vznikne tvar (4.38). Ten lze reprezentovat v maticovém tvaru (4.39) jako $MA_k = 0$ [2].

$$R(l) = \sum_{n=-\infty}^{\infty} x(n)x(n+l) \tag{4.37}$$

$$\sum_{i=0}^{k} a_i R(|j-i|) = 0, \quad j = 1, \ldots, k \tag{4.38}$$

$$A_k = \begin{bmatrix} 1 \\ a_1 \\ a_2 \\ \vdots \\ a_k \end{bmatrix}, \quad M = \begin{bmatrix} R(1) & R(0) & R(1) & \ldots & R(k-1) \\ R(2) & R(1) & R(0) & \ldots & R(k-2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ R(k-1) & R(k-2) & \ldots & R(2) & R(1) \\ R(k) & R(k-1) & \ldots & R(1) & R(0) \end{bmatrix} \tag{4.39}$$

K vyřešení soustavy se používá Levinson-Durbinův algoritmus. Ten řeší soustavy s maticí, která je symetrická a Töplitzova (prvky na diagonálách jsou stejné) [4]. Algoritmus vyhodnocuje koeficienty postupně a je založen na faktu, že je snadné spočítat koeficienty pro velikost $k+1$ pokud je známé řešení pro velikost k. Uvedený maticový zápis lze převést na tvar (4.40), kde matice N_k vyhovuje podmínce, že je symetrická a Töplitzova [2]. Energii chyby E_k lze popsat také vzorcem (4.41) [4].

$$N_k A_k = \begin{bmatrix} E_k \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad N_k = \begin{bmatrix} R(0) & R(1) & \dots & R(k) \\ R(1) & R(0) & \dots & R(k-1) \\ \vdots & \vdots & \ddots & \vdots \\ R(k) & R(k-1) & \dots & R(0) \end{bmatrix}, \quad A_k = \begin{bmatrix} 1 \\ a_1 \\ a_2 \\ \vdots \\ a_k \end{bmatrix} \tag{4.40}$$

$$E_k = R(0) + \sum_{i=1}^{k} a_i R(i) \tag{4.41}$$

Nejprve se vypočítá krok pro $k = 1$ a získá koeficient a_1 (výsledek (4.43)) a energie chyby E_1 (výsledek (4.44)) ze soustavy (4.42).

$$\begin{bmatrix} R(0) & R(1) \\ R(1) & R(0) \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \end{bmatrix} = \begin{bmatrix} E_1 \\ 0 \end{bmatrix} \tag{4.42}$$

$$a_1 = -\frac{R(1)}{R(0)} \tag{4.43}$$

$$E_1 = R(0) + R(1)a_1 \tag{4.44}$$

Další postup je stejný pro každý krok $k+1$. Vektor A_k se rozšíří o jeden řádek s nulou a označí U_{k+1}. Soustava má pak tvar (4.45). Protože je matice N_{k+1} symetrická, obrácením pořadí řádků vektoru U_{k+1} (označeno V_{k+1}) vznikne (4.46). Lineární kombinace (4.50) má tvar vektoru A_{k+1} s jedničkou na prvním řádku. Ze soustavy (4.47) lze vyjádřit λ (rovnice (4.49)) podle vztahu (4.48) [2].

$$\begin{bmatrix} R(0) & R(1) & \dots & R(k+1) \\ R(1) & R(0) & \dots & R(k) \\ \vdots & \vdots & \ddots & \vdots \\ R(k+1) & R(k) & \dots & R(0) \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ a_2 \\ \vdots \\ a_k \\ 0 \end{bmatrix} = \begin{bmatrix} E_k \\ 0 \\ 0 \\ \vdots \\ 0 \\ \sum_{j=0}^{k} a_j R(k+1-j) \end{bmatrix} \tag{4.45}$$

$$\begin{bmatrix} R(0) & R(1) & \dots & R(k+1) \\ R(1) & R(0) & \dots & R(k) \\ \vdots & \vdots & \ddots & \vdots \\ R(k+1) & R(k) & \dots & R(0) \end{bmatrix} \begin{bmatrix} 0 \\ a_k \\ \vdots \\ a_2 \\ a_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \sum_{j=0}^{k} a_j R(k+1-j) \\ 0 \\ \vdots \\ 0 \\ 0 \\ E_k \end{bmatrix} \tag{4.46}$$

$$N_{k+1} \begin{bmatrix} 1 \\ a_1 + \lambda a_k \\ a_2 + \lambda a_{k-1} \\ \vdots \\ a_k + \lambda a_1 \\ \lambda \end{bmatrix} = \begin{bmatrix} E_k + \lambda \sum_{j=0}^{k} a_j R(k+1-j) \\ 0 \\ 0 \\ \vdots \\ 0 \\ \sum_{j=0}^{k} a_j R(k+1-j) + \lambda E_k \end{bmatrix} \tag{4.47}$$

$$\sum_{j=0}^{k} a_j R(k+1-j) + \lambda E_k = 0 \tag{4.48}$$

$$\lambda = \frac{-\sum_{j=0}^{k} a_j R(k+1-j)}{E_k} \tag{4.49}$$

$$A_{k+1} = U_{k+1} + \lambda V_{k+1} \tag{4.50}$$

$$E_{k+1} = E_k + \lambda \sum_{j=0}^{k} a_j R(k+1-j) = (1-\lambda^2) E_k \tag{4.51}$$

Výsledný algoritmus výpočtu koeficientů se dá zapsat následovně [2].

- volba počtu koeficientů m
- výpočet $R(j), \quad j = 0 \dots m$ podle (4.37)
- výpočet A_1 podle (4.43)
- výpočet E_1 podle (4.44)
- pro $k = 1 \dots m$
  - výpočet λ podle (4.49)
  - výpočet U_{k+1}, V_{k+1}, A_{k+1} podle (4.50)
  - aktualizace E_{k+1} podle (4.51)

5 Vývojové nástroje

Pro vývoj zvukových aplikací a VST pluginů existuje řada specializovaných nástrojů a frameworků pro různé platformy i programovací jazyky. Ty se objevují v komerčních i nekomerčních a open-source verzích. Výhodou jejich použití je jednodušší a rychlejší vývoj, protože obvykle obsahují podpůrné programy a knihovny, které řeší typické problémy návrhu a vývojář se tak může soustředit hlavně na konkrétní zadání.

5.1 JUCE

Framework JUCE[1] je projekt společnosti ROLI[2] vyvíjený v jazyce `C++` pro tvorbu multiplatformních zvukových aplikací a pluginů pro počítače i mobilní zařízení. Projekt JUCE je částečně open-source a je k dispozici v několika verzích pro použití komerčně, zdarma i pro vzdělávací účely. Disponuje velkou uživatelskou základnou s vlastním diskuzním fórem a detailní dokumentací.

JUCE obsahuje knihovny s třídami a funkcemi určenými pro zpracování zvukového signálu a tvorbu grafického uživatelského rozhraní. Minimalizuje tak nutnost používat další knihovny třetích stran při vývoji aplikací. Dále podporuje vícevláknové programování, práci s grafikou apod.

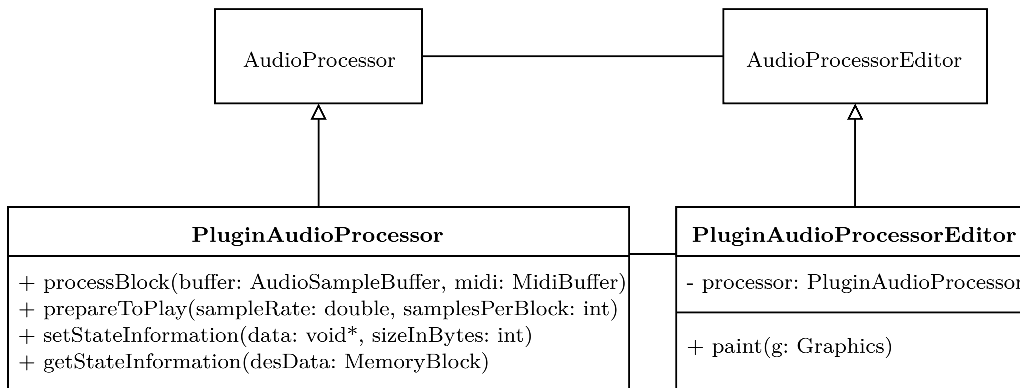
Základní myšlenkou frameworku je, aby programátor psal multiplatformní kód, který se soustředí pouze na vlastní funkci a ovládání pluginu. Ten by měl jít přeložit pro různé standardy (VST, RTAS, AU, AAX) a platformy (Windows, Mac OS X, Linux, iOS, Android).

Součástí JUCE je vývojové prostředí Projucer pro správu a vývoj projektů. Projucer funguje jako průvodce při vytvoření a nastavení konkrétního typu projektu. Po nastavení vygeneruje projekt obsahující knihovní třídy a šablony tříd pro vlastní kód. Projucer obsahuje vlastní textový editor a editor pro tvorbu grafického uživatelského rozhraní. Dokáže vygenerovat projekt pro překlad a vývoj v prostředí pomocí Microsoft Visual Studio, Xcode, Linux Makefile, Android Ant, Code::Blocks a CLion.

[1] `https://juce.com`
[2] `https://roli.com`

5.1.1 Projekt VST pluginu v JUCE

Po vytvoření projektu zvukového pluginu v Projucer jsou v adresáři se zdrojovými kódy třídy `PluginAudioProcessor` a `PluginAudioProcessorEditor` (závisí na názvu projektu). Diagram tříd je znázorněn na obrázku 5.1, kde jsou vyznačeny jen popisované proměnné a metody. Podle konvencí jazyka `C++` je deklarace v souborech s příponou `.h` oddělená od implementace v souborech `.cpp`.



Obrázek 5.1: Diagram tříd efektu v JUCE

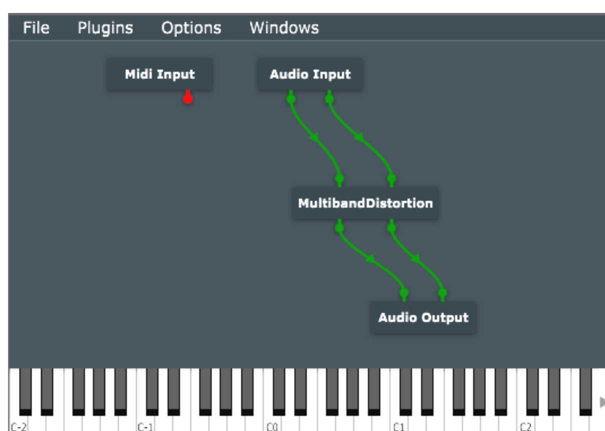
Třída `PluginAudioProcessor` je potomkem třídy `AudioProcessor` z frameworku JUCE. Vlastní implementace tak přepisuje metody rodičovské třídy pro zpracování zvuku a fungování pluginu. Veškeré zpracování zvukových dat probíhá v metodě `processBlock`. Ta se volá při předání bloku zvukových dat a MIDI událostí hostitelskou aplikací. Parametry metody jsou reference na instance třídy `AudioSampleBuffer` se zvukovými daty a `MidiBuffer` s MIDI událostmi. Při změně nastavení zvukových zařízení, jako je například změna vzorkovací frekvence, se volá metoda pro obsluhu změny parametrů zpracování `prepareToPlay`. Parametry předávané metodě jsou vzorkovací frekvence a maximální velikost bloku v počtu vzorků zvukových dat. Tato metoda se volá alespoň jednou při inicializaci pluginu a je užitečná při použití algoritmů pro zpracování, které jsou závislé na těchto parametrech, jako jsou například filtry.

Zvukové pluginy v hostitelských aplikacích mohou ukládat svůj stav. To je užitečné pro zachování nastavení pluginu při uložení projektu v DAW. Pro uložení stavu slouží metoda `getStateInformation` a pro obnovení slouží metoda `setStateInformation`. Hostitelská aplikace při spuštění a ukládání poskytne blok paměti pro zápis aktuálního a čtení předchozího stavu.

Třída `PluginAudioProcessorEditor` je potomek třídy `AudioProcessorEditor`. Jejím účelem je vytvoření grafického uživatelského rozhraní. Zákla-

dem je metoda `paint`, která vytváří okno editoru a umisťuje ovládací prvky. Zároveň může fungovat jako posluchač událostí ovládacích prvků a měnit parametry procesoru voláním metod instance třídy `PluginAudioProcessor` pro změnu parametrů. Pro tento účel má editor referenci na instanci třídy procesoru jako vlastní atribut.

Projekt obsahuje kromě těchto tříd ještě knihovní třídy JUCE. Z projektu lze přeložit plugin pro hostitelskou DAW aplikaci nebo jako samostatnou spustitelnou aplikaci. Součástí frameworku jsou i ukázkové aplikace, mezi které patří Juce Plug-in Host (obrázek 5.2). Po jejím přeložení vznikne spustitelný soubor, který obsahuje grafické rozhraní pro testování pluginů bez potřeby plnohodnotné DAW aplikace.



Obrázek 5.2: Juce Plug-in Host

5.2 WDL-OL

WDL-OL[3] je bezplatný open-source `C++` framework pro vývoj multiplatformních zvukových pluginů. Je to rozšířená verze frameworku IPlug společnosti Cockos[4], který byl původně součástí frameworku WDL. Stejně jako u JUCE je jeho hlavní myšlenkou použití stejného funkčního kódu k vytvoření pluginu pro různé platformy (Windows, Mac OS X) ve formátech VST, AU, RTAS a AAX. Protože je WDL-OL postaveno na frameworku WDL, jsou jeho součástí i některé základní funkce a algoritmy pro zpracování zvukového signálu, podobně jako v JUCE.

Součástí frameworku je jeden ukázkový projekt, který slouží jako šablona pro tvorbu vlastních projektů. Na rozdíl od frameworku JUCE, jehož sou-

[3] `https://github.com/olilarkin/wdl-ol`
[4] `https://www.cockos.com`

částí je prostředí Projucer, obsahuje WDL-OL pouze skript v jazyce Python pro vytvoření vlastního projektu. Projekt obsahuje knihovní kód a soubory pro vlastní funkci pluginu. Mezi ně patří hlavičkový soubor `resource.h`, který obsahuje definice konstant jako je jméno, ID, zdroje obrázků, velikost okna editoru apod.

Dalším rozdílem oproti JUCE je, že nastavení grafického rozhraní i zpracování zvukových dat je v jedné třídě, přičemž vykreslení ovládacích prvků se definuje v konstruktoru. Kód této třídy je v souborech `Plugin.h` a `Plugin.cpp` (pojmenování třídy závisí na zvoleném názvu projektu). Třída je potomkem knihovní třídy `IPlug` a podobně jako v JUCE implementuje několik základních metod pro zpracování signálu, a to `Reset`, která se volá při změně vzorkovací frekvence, `OnParamChange` pro změnu hodnoty parametru z grafického rozhraní a `ProcessDoubleReplacing`, která je jádrem pluginu a stejně jako `processBlock` v JUCE obsahuje kód pro zpracování zvukového signálu. Seznam parametrů se definuje jako výčtový typ a při změně hodnoty ovládacího prvku se odkazuje na parametr pomocí indexu.

5.3 RackAFX

RackAFX[5] je další volně dostupný nástroj pro vývoj multiplatformních VST a AU pluginů pro Windows a Mac OS X. Má vlastní prostředí pro založení projektu, editaci grafického uživatelského rozhraní a testování pluginu. V porovnání s JUCE tak kombinuje výhody programů Projucer a Juce Plug-In Host a poskytuje nástroje pro testování a návrh, jako je například možnost načíst a spustit zvukový záznam nebo návrh filtrů.

Po založení projektu a definici ovládacích prvků v grafickém rozhraní RackAFX následuje možnost vytvořit AU nebo VST plugin. V závislosti na platformě se vytvoří projekt (např. Visual Studio Solution). Ten podobně jako u předchozích obsahuje vygenerovaný kód s třídou `Plugin` (závisí na jménu projektu) určenou ke zpracování zvuku. Ta dědí od abstraktní třídy `CPlugIn` frameworku RackAFX. Podobně jako u předchozích nástrojů implementuje řadu metod včetně základních pro nastavení parametrů `prepareForPlay` a zpracování bloku zvukových dat poskytnutých hostitelem `processAudioFrame`.

[5] `http://www.willpirkle.com`

5.4 VST.NET

VST.NET[6] je open-source framework pro vývoj VST aplikací v prostředí .NET. Na rozdíl od předchozích se zaměřuje pouze na vývoj VST pluginů a hostitelských aplikací v jazyce `C#`. Neposkytuje žádné grafické rozhraní ani jinou automatizovanou možnost založení a konfigurace vlastního projektu. VST.NET se skládá ze tří DLL knihoven `Framework`, `Core` a `Interop`, které obsahují kód a rozhraní pro interoperabilitu mezi spravovaným kódem v `C#` a nespravovaným kódem v podobě volání funkcí API VST napsaných v `C` a `C++`.

Pro funkci pluginu je potřeba implementovat několik tříd. První je implementace abstraktní třídy `StdPluginCommandStub` a její metody `CreatePluginInstance`, která vrací instanci potomka třídy `VstPluginBase`. Ta sdružuje instance tříd pro zpracování signálu, nastavení parametrů a ukládání stavu. Vlastní zpracování probíhá ve třídě, která dědí od `VstPluginAudioProcessorBase` a přepisuje atribut pro nastavení vzorkovací frekvence `SampleRate` a metodu `Process` pro zpracování bloku dat předávaných hostitelskou aplikací podobně, jako je tomu u předchozích frameworků.

Jedním z dalších rozdílů oproti předchozím frameworkům je to, že přeložený plugin je potřeba distribuovat DAW aplikaci společně s DLL knihovnou `Interop`. Framework VST.NET neposkytuje žádné dodatečné funkce a algoritmy používané pro zpracování zvuku, jako je tomu například u JUCE nebo WDL-OL.

5.5 Shrnutí

Struktura kódu pro zpracování zvukových dat se mezi nástroji příliš neliší. Všechny projekty v těchto nástrojích implementují jednu nebo více tříd s metodami pro zpracování bloku zvukových dat nebo MIDI zpráv, změnu parametrů a nastavení uživatelského rozhraní, a odpovídají tak přibližně rozhraním z SDK VST.

Rozdíly jsou v podporovaných platformách a formátech zvukových pluginů, dokumentaci, podpoře a dalších funkcích a nástrojích. Nejméně možností poskytuje VST.NET, který umožňuje vytvoření pluginu pouze jako VST a neobsahuje žádné knihovní funkce zpracování zvukového signálu. WDL-OL a RackAFX umožňují multiplatformní vývoj pluginů v různých formátech a mají knihovní funkce a nástroje pro snazší a rychlejší návrh pluginu. Nástroj JUCE je jediný, který je k dispozici i v komerčních verzích.

[6] `https://github.com/obiwanjacobi/vst.net`

Jeho výhodou je podpora ve formě vlastního diskuzního fóra, detailní dokumentace a množství návodů na jeho webových stránkách. JUCE je stále aktivně vyvíjen a kromě nástrojů pro snazší návrh a ladění pluginu jsou k dispozici obsáhlé knihovny funkcí používaných při zpracování zvukového signálu, a proto byl JUCE vybrán pro implementaci vlastního efektu.

6 Open-source pluginy

Vedle komerčních VST pluginů a těch, které jsou součástmi aplikací DAW, existuje i řada open-source implementací efektů dostupných na Internetu. Ty se vyskytují ve formě plnohodnotných efektů nebo jako vzorové příklady použití SDK VST a vývojových nástrojů.

6.1 mda-vst

Mda-vst[1] je sada třiceti efektů a čtyř nástrojů distribuovaných jako open-source. Zdrojové kódy jsou jako příklady součástí SDK VST a jsou napsány v jazyce `C++`.

Každý plugin je tvořen dvěma třídami `Processor` a `Controller` s doplněným názvem efektu na začátku. Všechny třídy jsou potomky základních tříd `BaseController` nebo `BaseProcessor`. Tyto základní třídy implementují rozhraní z SDK VST popsané v kapitole 3 a implementují společnou funkcionalitu, která je konkrétními pluginy přepisována. Třída `BaseController` implementuje rozhraní kontroleru a stará se tak o ovládání pluginu. Abstraktní třída `BaseProcessor` implementuje rozhraní procesoru. Jedná se o metody `process` pro zpracování bloku zvukových dat a MIDI zpráv, metody pro ukládání a načtení stavu pluginu `setState` a `getState`, dále pak metody pro nastavení maximální velikosti bloku, aktivaci a deaktivaci pluginu a metodu `setBusArrangements`, která vynucuje nastavení přesně dvou vstupních a dvou výstupních kanálů od hostitelské aplikace. Třída definuje vlastní virtuální metody pro nastavení a změnu parametrů, jejich alokaci apod. Dále má jedinou čistě virtuální metodu `doProcessing` pro implementaci konkrétního algoritmu pluginu a je volána z metody `process`.

Mda-vst obsahuje i efekty používající postupy popsané v předchozích kapitolách. Jedním z nich je efekt zkreslení Overdrive. Má tři parametry, a to úroveň zkreslení *drive* v rozsahu $0 - 100$ %, *muffle* pro nastavení tónu v rozsahu $0 - 100$ % a *output* pro nastavení výstupní hlasitosti v rozsahu od -60 do 0 dB. Algoritmus zkreslení postupně aplikuje na všechny vzorky zkreslení podle vztahu (6.1), dolní propust podle parametru *muffle* a nakonec

[1] `http://mda.smartelectronix.com`

zesílení podle parametru *output*.

$$f(x) = \begin{cases} \sqrt{x}, & x > 0 \\ -\sqrt{-x}, & x < 0 \\ 0, & x = 0 \end{cases} \tag{6.1}$$

Dalším zkreslením v této sadě je efekt Bandisto. Bandisto je příkladem vícepásmového efektu. Konkrétně používá tři frekvenční pásma s nastavitelnou šířkou a vlastním zkreslením. Každému pásmu se dá nastavit míra zkreslení jako zesílení v rozsahu od 0 do 60 dB a hlasitost výstupu od -20 do 20 dB. Pro všechna zkreslení existuje možnost volby mezi bipolárním a unipolárním režimem zkreslení. Rozdíl je ve způsobu změny tvaru vrcholů vlny, kde bipolární režim mění tvar vrcholů vlny v maximech i minimech a unipolární jen v maximech nebo minimech. Na každý vzorek v metodě `doProcessing` se aplikuje filtr a následně se zkreslí podle vztahu (6.2), kde d je zvolená míra zkreslení a t zvolená výstupní hlasitost pro dané frekvenční pásmo.

$$f(x) = \frac{t \times x}{1 + d \times |x|} \tag{6.2}$$

V případě unipolárního režimu se dá zkreslení jednoho filtrovaného vzorku zapsat vztahem (6.3). Všechny tři filtrované a upravené vzorky jednoho kanálu se na výstupu sečtou.

$$g(x) = \begin{cases} t \times x, & x >= 0 \\ f(x), & x < 0 \end{cases} \tag{6.3}$$

Sada mda-vst obsahuje také efekt zpoždění Delay. Nastavení zpoždění je možné u obou kanálů zvlášť a dochází tak ke stereo efektu. U levého se nastavuje od 0 do 742 milisekund. Pravý kanál se určuje jako procento zpoždění levého kanálu. Procesor uchovává jednu cyklickou vyrovnávací paměť. V každém průchodu se nejdříve přečte aktuální vzorek levého x_L a pravého kanálu x_R. Z vyrovnávací paměti se přečtou zpožděné vzorky pro levý d_L a pravý d_R kanál podle parametrů jejich zpoždění podobně jako je popsáno v kapitole 4. Vypočítá se vzorek x (6.4), který je filtrován dolní propustí na základě nastavení parametru *tone* a uložen na další pozici ve vyrovnávací paměti.

$$x = w \times (x_L + x_R) + f \times (d_L + d_R) \tag{6.4}$$

Parametry *wet* (w – popsáno v kapitole 4) a *feedback* (f) jsou nastavitelné ovládacími prvky s rozsahem $0 - 100$ %. Parametr *wet* určuje poměr

s parametrem *dry* (d). Na konec se vypočítají hodnoty vzorků pro oba výstupní kanály y_L a y_R podle (6.5).

$$\begin{aligned} y_L &= d \times x_L + d_R \\ y_R &= d \times x_R + d_R \end{aligned} \tag{6.5}$$

6.2 ADelay

Efekt ADelay je další z efektů, které jsou součástí SDK VST. Na rozdíl od efektu Delay ze sady *mda-vst* má pouze jeden parametr zpoždění, a tak jsou všechny kanály zpožděné stejně. Dalším rozdílem je, že efekt neumožňuje tzv. *dry/wet mix*, takže výstupem je čistý původní signál zpožděný o určenou dobu. V implementaci se v metodě `setBusArrangements` vynucuje, aby byl hostitelem poskytnut stejný počet vstupních a výstupních kanálů. Každý kanál musí mít vlastní cyklickou vyrovnávací paměť implementovanou jako pole a proměnné s aktuálním indexem pro ukládání aktuálních a čtení zpožděných vzorků signálu. Počet vzorků ke zpoždění se získá jako násobek vzorkovací frekvence a parametru zpoždění v sekundách.

6.3 Audio Effects

Součástí doprovodných materiálů ke knize *Audio Effects: Theory, Implementation and Application* od autorů Joshuy D. Reisse a Andrewa P. McPhersona jsou implementace zvukových efektů, které jsou volně k dispozici[2]. Sada obsahuje osmnáct efektů realizovaných jako VST pluginy ve frameworku JUCE s vlastním uživatelským rozhraním, z nichž některé jsou v následujícím textu popsány.

Prvním zkoumaným pluginem je efekt zkreslení. Kód je tvořen pro JUCE typicky dvěma třídami `DistortionAudioProcessor` a `DistortionAudioProcessorEditor`. Má dva ovládací prvky, a to zesílení *gain*, které se aplikuje vždy před zkreslením, a výběr typu zkreslení. Prvním typem je tzv. *hard-clipping* v podobě, jak je popsaný v kapitole 4 s pevně danou prahovou hodnotou 0,5. Další jsou dvě varianty tzv. *soft-clippingu*. První způsob se dá

[2]`https://code.soundsoftware.ac.uk/projects/audio_effects_textbook_code`

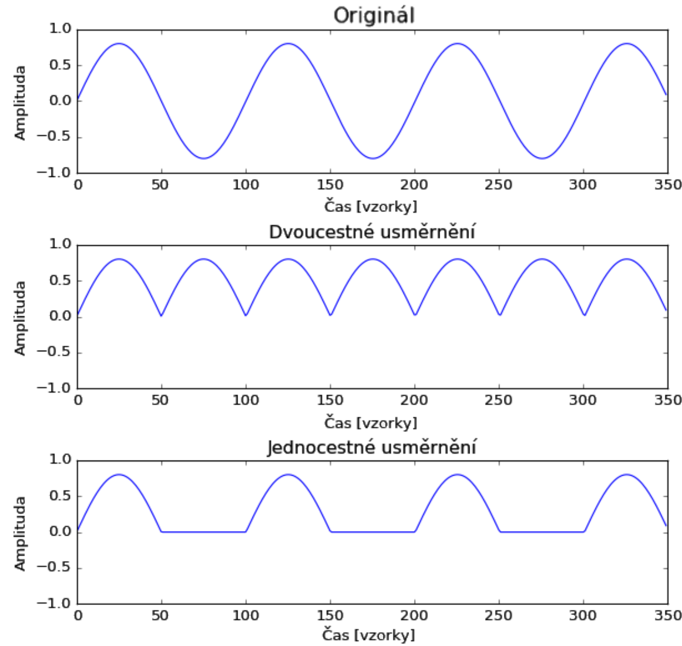
zapsat funkcí (6.6), druhý způsob je podle funkce (4.11) z kapitoly 4.

$$f(x) = \begin{cases} 1, & x > \frac{1}{3} \land x > \frac{2}{3} \\ \frac{3-(2-3x)(2-3x)}{3}, & x > \frac{1}{3} \land x \leq \frac{2}{3} \\ -1, & x < -\frac{1}{3} \land x < -\frac{2}{3} \\ -\frac{3-(2+3x)(2+3x)}{3}, & x < -\frac{1}{3} \land x \geq -\frac{2}{3} \\ 2x, & x \geq -\frac{1}{3} \land x \leq \frac{1}{3} \end{cases} \tag{6.6}$$

Poslední dva typy zkreslení jsou založené na principu usměrňovače. První je dvoucestné usměrnění, kde každý výstupní vzorek je absolutní hodnotou vstupního vzorku (6.7). Druhý typ je jednocestné usměrnění, které na výstupu nechává jen vzorky větší než nula (6.8) a ostatní nastaví na nulu. Výsledek efektů je znázorněn na obrázku 6.1.

$$f(x) = |x| \tag{6.7}$$

$$f(x) = \frac{|x| + x}{2} \tag{6.8}$$

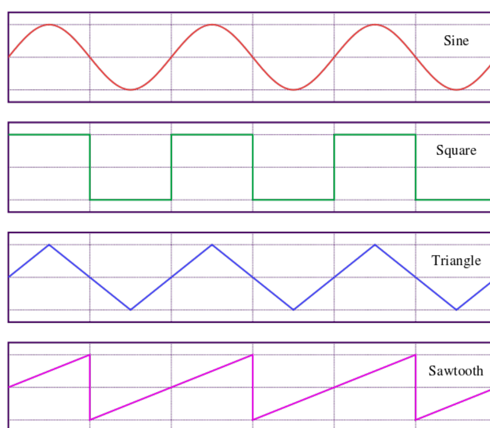


Obrázek 6.1: Zkreslení usměrněním

Součástí této sady je také efekt zpoždění. Nastavitelné parametry efektu jsou zpoždění v sekundách, zpětná vazba (*feedback*) a *dry/wet mix* v rozsahu od 0 do 1. V implementaci se počítá s více vstupními kanály, ale cyklická

vyrovnávací paměť se používá jen jedna. Algoritmus je založen na tom, že velikost bloků je v jednom volání processBlock pro všechny kanály stejná. Ukazatele pro čtení a zápis do vyrovnávací paměti se před zpracováním vzorků v bloku uloží do lokální proměnné a po zpracování všech kanálu se nastaví jejich aktuální hodnota. Díky těmto lokálním kopiím se pracuje u každého kanálu s vyrovnávací pamětí stejně. Myšlenka výpočtu výstupního vzorku je stejná, jako je uvedeno ve vztahu (4.12) v kapitole 4. Jako další vzorek do vyrovnávací paměti se ukládá součet aktuálního vstupního vzorku a násobku zpětné vazby se zpožděným vzorkem.

Efekty *tremolo* a *vibrato* založené na modulaci pomocí LFO jsou k dispozici v sadě také. Tremolo nabízí možnost nastavení frekvence oscilace LFO od 0,2 do 20 Hz, minimální hloubku změny amplitudy v rozsahu $0 - 1$ a výběr tvaru generované vlny mezi sinusoidou, trojúhelníkovou a obdélníkovou vlnou (obrázek 6.2). Pro generování vlny pomocí LFO je potřeba ukládat stav mezi voláním metody processBlock tak, aby byla vlna spojitá. K tomu se ukládá fáze jako atribut třídy, která se před zpracováním každého kanálu uloží do lokální proměnné a po zpracování všech kanálů se aktualizuje. Tento způsob je opět založen na faktu, že všechny kanály v jednom volání processBlock mají stejnou velikost bloku dat.



Obrázek 6.2: Porovnání obdélníkové (zeleně), trojúhelníkové (modře) a pilové (fialově) vlny se sinusoidou
Zdroj: https://commons.wikimedia.org

U efektu *vibrato* je možné nastavit rozsah vlnění v sekundách a frekvence vlnění generovaného LFO. Stejně jako u předchozího efektu je potřeba zachovávat fázi vlny mezi voláním metody pro zpracování bloku. Efekt je dosažen změnou zpoždění, jak je popsáno v kapitole 4. Navíc je možné zvolit tvar generované vlny jako sinusoidu, trojúhelníkovou nebo pilovou vlnu. Další do-

datečnou funkcí u tohoto efektu je možnost zvolit způsob interpolace vzorků. Způsob nalezení zpožděného vzorku z kapitoly 4 ve vyrovnávací paměti najde vzorek s nejbližším indexem. Efekt *vibrato* z této sady nabízí možnost lineární a kubické interpolace vzorku. To umožňuje hladší průběh změny frekvence, ale způsobuje vyšší výpočetní náročnost.

7 Implementace efektů v JUCE

K implementaci vlastního navrženého efektu byl vybrán nástroj JUCE z důvodů uvedených v kapitole 5. Pro seznámení se ním a prozkoumání některých základních technik bylo před implementací vlastního efektu vyvinuto pět jednoduchých efektů, které jsou součástí této práce.

7.1 Distortion

Efekt zkreslení byl vytvořen jako první vzhledem k jednoduchosti jeho implementace. Jako algoritmus zkreslení zvuku je použit tzv. *hard-clipping* popsaný v kapitole 4. Efekt se ovládá dvěma parametry pro zesílení signálu (*gain*) a nastavení prahové hodnoty (*threshold*). Výpis kódu 7.1 z metody `processBlock` znázorňuje jádro efektu, kde se na každý vzorek každého vstupního kanálu aplikuje zkreslení.

```
1  float* channelData = buffer.getWritePointer (channel);
2
3  for (int i = 0; i < buffer.getNumSamples(); i++) {
4    channelData[i] *= gain;
5
6    if (channelData[i] > threshold)
7      channelData[i] = threshold;
8
9    if (channelData[i] < -threshold)
10     channelData[i] = -threshold;
11 }
```

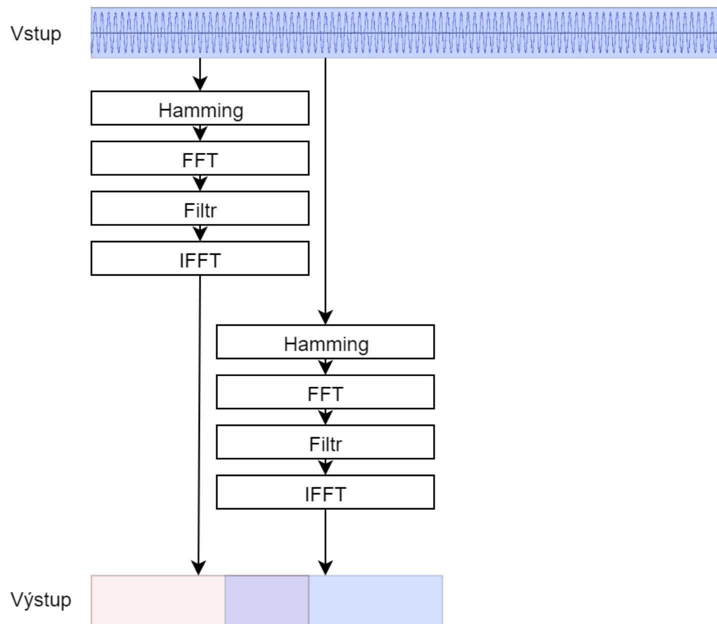
Výpis kódu 7.1: Zkreslení v JUCE

7.2 Filter

S použitím metody STFT a její inverze byl vytvořen plugin jednoduchého frekvenčně selektivního filtru. Tento plugin má dva parametry, a to zlomové frekvence pro dolní a horní propust v rozsahu od 0 do 20 kHz. Filtr se tak dá použít jako dolní, horní i pásmová propust. Návrh odpovídá způsobu, který je popsaný v kapitole 4 s velikostí analyzovaného bloku $N = 512$ vzorků

a polovičním posuvem $H = 256$ vzorků. Pro vyhlazení bloků je použito Hammingovo okno.

Protože velikost bloku předávaná hostitelskou aplikací se bude pravděpodobně lišit od velikost bloku určenému k analýze, je potřeba ukládat vstupní vzorky do vyrovnávacích pamětí, které jsou pro každý kanál samostatné. Stejně tak je potřeba připravit vyrovnávací paměť pro každý výstupní kanál, odkud se budou číst filtrované vzorky. Pro tento účel je v projektu kromě zdrojových kódu procesoru a editoru také vlastní implementace cyklické vyrovnávací paměti ve třídě `CyclicBuffer`.



Obrázek 7.1: FFT filtrování

Algoritmus je znázorněn obrázkem 7.1. Po naplnění vstupní vyrovnávací paměti se její data vynásobí vyhlazovacím oknem, provede se FFT, následně filtrování a inverzní FFT. Filtrování se provede nastavením nulové hodnoty na pozicích frekvenčních pásem pod zlomovou frekvencí dolní propusti a nad zlomovou frekvencí horní propusti, a odpovídá tak násobení obdélníkovou funkcí ve frekvenční doméně. Tento způsob filtrování s polovičním překrytím způsobuje zpoždění o $\frac{N}{2}$ vzorků a kvůli ostrým přechodům mezi pásmy ve frekvenční doméně může způsobovat šum ve výstupním signálu při filtrování některých frekvencí.

7.3 Delay

Delay má dva parametry, kterými jsou zpoždění v sekundách a poměr směšování mezi aktuálním a zpožděným signálem. Implementace zachovává principy popsané v předchozích kapitolách.

Je implementována vlastní cyklická vyrovnávací paměť jako třída `DelayBuffer`. Té se v konstruktoru nastavuje vzorkovací frekvence a maximální zpoždění v sekundách. Na základě toho se alokuje vektor pro ukládání vzorků. Třída má dvě metody `addSample` pro přidání dalšího vzorku a `getDelayedSample`, která vrací zpožděný vzorek na základě zpoždění v sekundách předávaného jako parametr. Pro každý vstupní kanál je jedna vyrovnávací paměť uložená ve vektoru `channelBuffers`. Zpracování jednoho kanálu je znázorněno ve výpisu kódu 7.2.

```
1  float* channelData = buffer.getWritePointer (channel);
2
3  for (int i = 0; i < buffer.getNumSamples(); i++) {
4    channelBuffers[channel]->addSample(channelData[i]);
5
6    delayedSample =
7        channelBuffers[channel]->getDelayedSample(delay);
8    channelData[i] = (1.f - wetMix) * channelData[i]
9        + wetMix * delayedSample;
10 }
```

Výpis kódu 7.2: Zpoždění v JUCE

7.4 Vibrato

Vibrato používá stejnou třídu `DelayBuffer` pro zpoždění signálu. Zpoždění není dáno pevně, ale osciluje od nuly k maximálnímu zvolenému. To se nastavuje jako parametr efektu v sekundách společně s frekvencí oscilace vlny generované LFO (Low Frequency Oscillator – popsáno v kapitole 4). Pro generování vlny slouží funkce `sinewave`, která vrací hodnotu sinusoidy podle amplitudy, času, frekvence a fáze. Pro spojitost generované vlny mezi voláním `processBlock` se na konci metody ukládá aktuální fáze jako atribut třídy. Fáze bude mezi voláním `processBlock` pro všechny kanály stejná, protože počet vzorků v jednom bloku je u všech kanálů stejný. Vlna generovaná LFO je posunutá o 0,5 s amplitudou 0,5, aby generovala hodnoty od 0 do 1. Čas v bloku se počítá pro každý kanál od nuly jako index vzorku násobený převrácenou hodnotou vzorkovací frekvence, která se nastavuje v metodě

`prepareToPlay`. Zpracování jednoho kanálu je znázorněno ve výpisu kódu 7.3.

```
1  float* channelData = buffer.getWritePointer (channel);
2
3  for (int i = 0; i < buffer.getNumSamples(); i++) {
4   channelBuffers[channel]->addSample(channelData[i]);
5
6   time = i * inverseSampleRate;
7   lfo = 0.5f + sinewave(0.5f, frequency, phase, time);
8
9   channelData[i] =
10 channelBuffers[channel]->getDelayedSample(lfo * delay);
11 }
```

Výpis kódu 7.3: Vibrato v JUCE

7.5 Tremolo

Tremolo periodicky mění amplitudu výstupu. Jeho parametry jsou hloubka (maximální velikost poklesu) amplitudy a frekvence vlny generované LFO. Maximální násobek amplitudy je vždy 1 a hloubka udává o jakou úroveň může amplituda klesnout. Stejně jako u efektu Vibrato se používá LFO a ukládá se fáze generované vlny po každém zpracování bloku. Na rozdíl od něj není potřeba uchovávat vzorky ve vyrovnávací paměti. Zpracování jednoho kanálu je naznačeno ve výpisu kódu 7.4

```
1  float* channelData = buffer.getWritePointer(channel);
2
3  for (int i = 0; i < buffer.getNumSamples(); i++) {
4    time = i * inverseSampleRate;
5    lfo = 0.5f + sinewave(0.5f, frequency, phase, time);
6
7    channelData[i] *= (1 - depth * lfo);
8  }
```

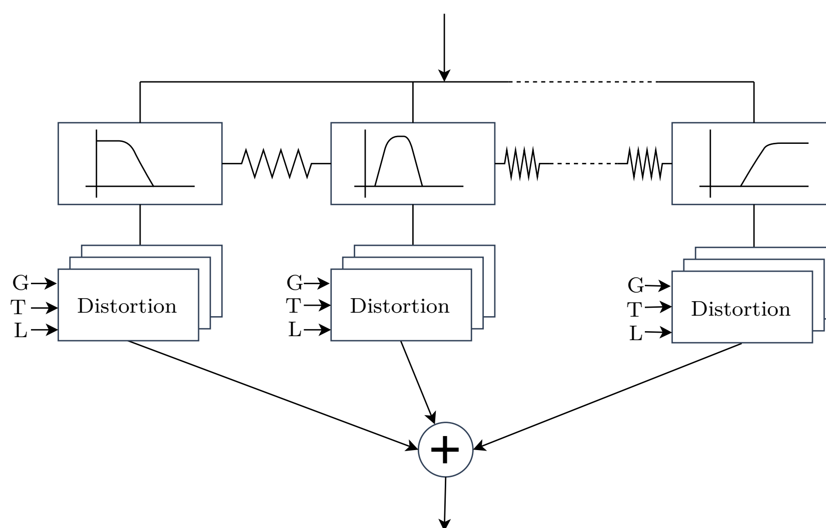
Výpis kódu 7.4: Tremolo v JUCE

8 Vícepásmové zkreslení

Jako výsledek této práce byl navržen a implementován efekt zkreslení ve formě VST pluginu s použitím frameworku JUCE pro nasazení v hostitelských DAW aplikacích, které podporují tento standard.

Efekt je navržen jako vícepásmový. To znamená, že se vstupní signál filtrací rozdělí do několika frekvenčních pásem, která se zkreslují samostatně a na výstupu se sloučí. Počet pásem je volitelný od dvou do pěti. Stejně tak je možné nastavit zlomové frekvence mezi sousedními pásmy, a určit tak jejich šířku. Každému pásmu je možné vybrat z nabídky různých algoritmů zkreslení, kterým se nastavují parametry (zesílení, prahová hodnota, výstupní hlasitost) samostatně. K dispozici je i vlastní navržený algoritmus zkreslení založený na lineární predikci, který adaptivně upravuje hladinu prahové hodnoty. Zlomovým frekvencím mezi pásmy je možné nastavit oscilaci. Na základě zvoleného rozpětí a frekvence oscilace se zlomové frekvence a šířka sousedních pásem periodicky mění.

Vícepásmový návrh s nastavitelným počtem a šířkou pásem, volitelným zkreslením a automatickou oscilací zlomových frekvencí poskytuje dostatečné možnosti k dosažení různých podob výsledného zvuku. Návrh efektu je schématicky znázorněn na obrázku 8.1.



Obrázek 8.1: Návrh efektu

8.1 Adaptivní zkreslení

Myšlenka adaptivního zkreslení je založená na změně způsobu zkreslení *hard-clipping* popsaného v kapitole 4. Prahová hodnota pro oříznutí vlny je konstantní a ke zkreslení zvuku dochází pouze v místech s amplitudou, která je vyšší a vstup je hlasitější. Aby ke zkreslení signálu docházelo rovnoměrně, je potřeba nejdříve signál zesílit tak, aby signál překračoval prahovou hodnotu, nebo prahovou hodnotu snížit, čímž dochází i ke snížení amplitudy výsledného signálu.

Adaptací prahové hodnoty na změnu amplitudy vzniká nový druh zkreslení, jehož intenzita se podle aktuálního vstupu mění. K tomuto přístupu je potřeba analyzovat průběh signálu před změnou prahové hodnoty. Možností je ukládat vzorky do vyrovnávací paměti a předávat je na výstup až po analýze a zpracování. Taková možnost ale způsobuje zpoždění na výstupu. Vzhledem k návrhu pluginu jako vícepásmového zkreslení by takové zpoždění mohlo být problematické, protože je možné kombinovat různé algoritmy zkreslení souběžně a bylo by nutné synchronizovat jejich zpoždění na stejnou hodnotu.

```
   input  : threshold, prediction order, size of block, array of samples
   output: samples
 1 tmpThreshold ← threshold;
 2 init buffer;
 3 foreach s in samples do
 4 |   add s to buffer;
 5 |   if s > tmpThreshold then
 6 |   |   s ← tmpThreshold;
 7 |   else if s < -tmpThreshold then
 8 |   |   s ← -tmpThreshold;
 9 |   end
10 |   if size of buffer = size then
11 |   |   maxValue ← PredictAbsMax (order, size, buffer);
12 |   |   tmpThreshold ← threshold × maxValue;
13 |   |   clear buffer;
14 |   end
15 end
```

Algoritmus 8.1: Adaptivní zkreslení

S použitím lineární predikce je možné odhadnout následující vzorky z předchozích. Odhad budoucího signálu dovoluje adaptovat prahovou hodnotu pro aktuální vzorky bez přidaného zpoždění. Základní myšlenkou algoritmu, která je naznačená v pseudokódu 8.1, je uložit blok vzorků do vyrovnávací paměti, z něj odhadnout stejný počet následujících vzorků, najít jejich absolutní maximum a podle něj určit prahovou hodnotu, která se bude aplikovat na následující blok. Velikost bloku a řád predikce (počet koeficientů pro odhad dalšího vzorku) je konstantní a měnitelným parametrem je pouze základ prahové hodnoty, ze které se určuje aktuální hodnota jako násobek odhadu absolutního maxima.

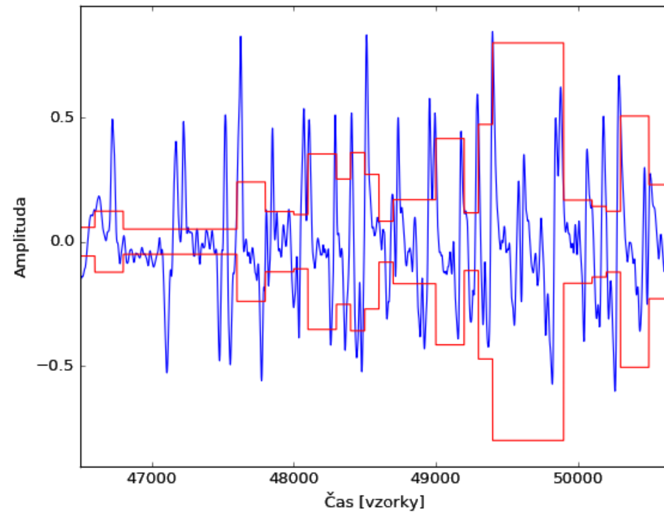
K odhadu absolutního maxima následujícího bloku vzorků se používá vztah (4.32) a algoritmus pro výpočet koeficientů lineární predikce z kapitoly 4. Koeficienty a_i se vypočítají na základě zvoleného řádu predikce a uloženého bloku vzorků. S jejich pomocí se z uloženého bloku predikuje stejně velký blok následujících vzorků. Odhad absolutního maxima vzorku je znázorněn pseudokódem 8.2.

```
input  : order, size, array of samples block
output: max
1  max ← −1;
2  coeffs ← LpcCoefficients (order, block);
3  blockSize ← size of block;
4  for i ← 0 to size do
5      predictedSample ← 0;
6      for j ← 0 to order do
7          predictedSample ← predictedSample − coeffs [j] × block
            [blockSize + i − 1 − j];
8      end
9      add predictedSample to block;
10     if |predictedSample| > max then
11         max ← |predictedSample|;
12     end
13 end
```

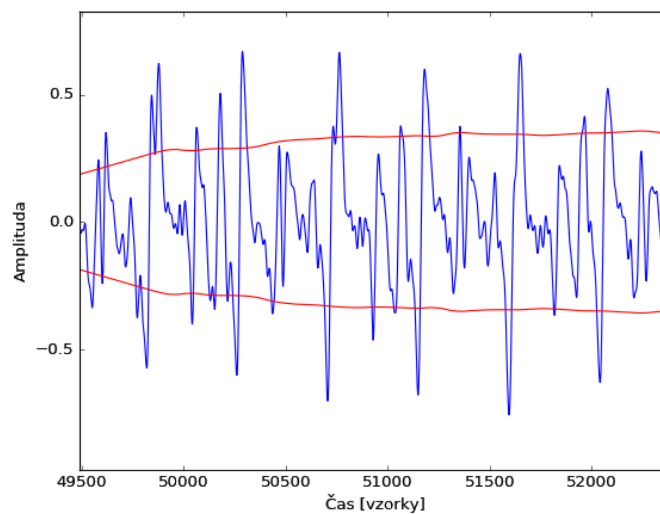
Algoritmus 8.2: Odhad absolutního maxima

Uvedený algoritmus zkreslení naznačuje základní myšlenku, ale je potřeba přidat několik vylepšení. Jak je vidět na obrázku 8.2, změny prahové hodnoty jsou skokové, protože nastavení se provádí pro celý následující blok

signálu. Výrazné změny prahové hodnoty jsou ve výsledném kódu vyhlazeny pomocí klouzavého průměru, kdy se ukládá pole několika posledních prahových hodnot a poslední hodnota se ukládá jako průměr. Úplného odstranění skoků je dosaženo kubickou interpolací, kdy se prahová hodnota interpoluje pro každý vzorek v bloku samostatně. Výsledek je znázorněn na obrázku 8.3.



Obrázek 8.2: Skokové změny prahové hodnoty



Obrázek 8.3: Vyhlazení změny prahové hodnoty

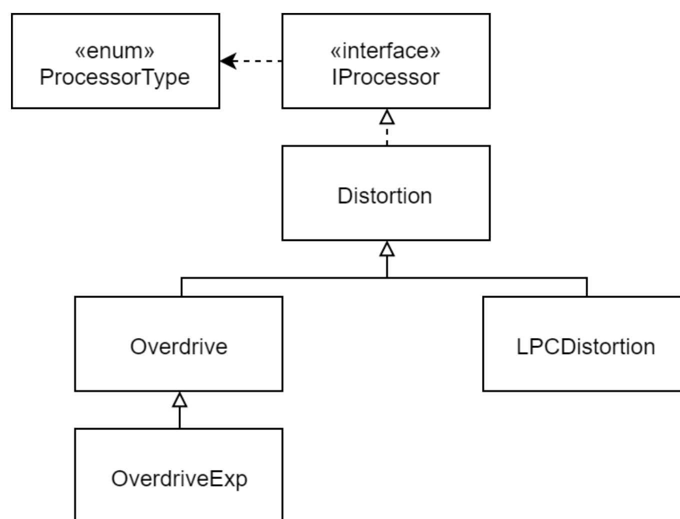
Hodnoty velikosti bloku vzorků pro predikci (100), řádu predikce (16), velikosti okna pro klouzavý průměr (32) a počet bodů pro interpolaci (32) jsou v implementaci zkreslení nastaveny konstantně a byly zvoleny testováním na základě pozorování průběhu prahové hodnoty a poslechem výsledného zvuku. Kromě nastavení základu prahové hodnoty se nastavuje zesílení

a úroveň výstupního signálu. Zesílení se aplikuje před zkreslením adaptovanou prahovou hodnotou, ale pro predikci se používá originální signál před zesílením. Výstupní úroveň se nastavuje při zpracování vzorku jako poslední.

Predikce velkého počtu vzorků ze stejného počtu skutečných vzorků způsobuje výraznou chybu. Odhad tak nemusí být přesný, ale přítomnost chyby může být v tomto případě považována za vlastnost, která vede k vytvoření originálního způsobu zkreslení. Protože adaptací může kolísat hlasitost, je každý vzorek po aplikaci aktuální prahové hodnoty touto hodnotou vydělený. Tím se rozsah výstupu dostane na hodnoty od -1 do 1. Při nastavování aktuální prahové hodnoty je ošetřeno, aby neklesla pod 0,05 a nepřiblížila se tak k nule. V pluginu je u každého typu zkreslení možné nastavit výstupní úroveň hlasitosti. Díky tomu se dá celková úroveň výstupu upravit.

8.2 Typy zkreslení

V pluginu jsou na výběr čtyři typy zkreslení. Pojmenovány jsou *Distortion* podle funkce (4.9), *Overdrive* podle funkce (4.10), *OverdriveExp* podle funkce (4.11) a *LPCDistortion* pro adaptivní zkreslení a každému pásmu lze nastavit jedno z nich. Všechny mají parametry *gain* pro zesílení v rozsahu od 0 do 50 dB a *level*. *Level* se nastavuje od -50 do 50 dB, slouží pro úpravu výstupních hlasitostí pásem a při hodnotách vyšších než nula může dojít k dalšímu zkreslení. Zkreslení typu *distortion* mají navíc nastavitelnou prahovou hodnotu od 0,1 do 1.



Obrázek 8.4: Diagram tříd zkreslení

Každý typ zkreslení je tvořen jednou třídou a implementuje stejné roz-

hraní `IProcessor` (obrázek 8.4). Rozhraní definuje metody pro nastavení a získání prahové hodnoty, zesílení, výstupní úrovně signálu, počtu kanálů a přepínání aktivního a neaktivního stavu. Protože návrh počítá s maximálně třemi parametry, které nemusí být u všech typů využity, definuje rozhraní metody pro zjištění, které z nich jsou funkční. Na základě toho se mění ovládací prvky v grafickém uživatelském rozhraní. Pro zpracování signálu slouží metoda `processSample`, jejíž parametry jsou reference na vzorek a číslo kanálu. Pro identifikaci typu zkreslení vrací metoda `getType` prvek výčtového typu `ProcessorType`. Instance třídy zkreslení tak uchovává nastavení jednoho pásma. Implementací rozhraní nebo děděním některé ze stávajících implementací lze přidat další typy zkreslení do pluginu.

8.3 Filtry

Každé pásmo v efektu je tvořeno instancí třídy implementující rozhraní `IProcessor` a instancí třídy filtru `FrequencyFilter`. Ty jsou sdruženy ve struktuře `ProcessingBand` a všechna pásma jsou v procesoru udržována jako vektor s prvky typu této struktury. Každý vzorek v metodě pro zpracování bloku vzorků signálu `processBlock` je zpracován všemi aktivními pásmy v tomto vektoru. U každého dojde nejdřív k nastavení zlomové frekvence filtru v případě nastavení oscilace, filtraci podle daného typu a nastavení filtru a následně ke zkreslení.

Oscilace mezi přechody pásem se nastavuje pomocí dvou parametrů, a to šířkou pásma a frekvencí. Stejně jako u efektů *tremolo*, kde se nastavuje amplituda a frekvence, a *vibrato*, kde se nastavuje maximální zpoždění a frekvence, se využívá generování vlny pomocí LFO. Nastavení zlomových frekvencí mezi pásmy je ve vektoru typu struktury `FrequencyCutoff`, kde se ukládá jejich hodnota, frekvence oscilace, rozsah oscilace a fáze. Aktuální hodnota zlomové frekvence f_c se vypočítá podle vztahu (8.1), kde F_c je zvolená dělící frekvence mezi pásmy, r rozsah oscilace, f frekvence oscilace, i index vzorku v bloku, f_s vzorkovací frekvence a φ je fáze, která se aktualizuje na konci volání `processBlock` po zpracování všech vzorků. Na rozdíl od efektů *tremolo* a *vibrato* je rozsah generovaných hodnot od $-0,5$ do $0,5$, aby hodnoty oscilovaly od zlomové frekvence o polovinu zvoleného rozsahu v obou směrech.

$$f_c = F_c + r\left(0,5\sin\left(2\pi f\frac{i}{f_s} + \varphi\right)\right) \tag{8.1}$$

Protože filtry si udržují stav a musí zpracovávat každý kanál samostatně,

byla vytvořena třída `FrequencyFilter`, aby obalovala konkrétní implementaci filtru. Jedna instance třídy patří jednomu pásmu pro zpracování signálu, ale vnitřně uchovává vektor filtrů podle počtu kanálů. Kromě počtu kanálů se nastavuje vzorkovací frekvence, spodní a horní zlomová frekvence a typ filtru. Typy použitých filtrů jsou dolní, horní a pásmová propust.

Nejdříve byla pro plugin použita vlastní implementace FIR filtru. Ta byla vhodná pro filtrování signálu do zvolených pásem, ale ukázalo se, že je neefektivní pro rychlé změny zlomových frekvencí způsobené oscilací přechodů mezi pásmy. Byla proto nahrazena třídou `dsp::StateVariableFilter`, která je součástí knihoven frameworku JUCE. Jedná se o IIR filtr navržený pro rychlou modulaci zlomových pásem [5]. Filtr se dá použít jako dolní, horní i pásmová propust, ale při testování se ukázalo, že při velké šířce pásma u pásmové propusti přestává fungovat a neprodukuje žádný výstup. Proto byla ve třídě `FrequencyFilter` implementována pásmová propust jako dvojice filtrů horní a dolní propusti.

Filtrování jednoho vzorku je zajištěno voláním metody `filterSample`. Jako parametr se předává číslo kanálu a hodnota vzorku. Během změny zlomových frekvencí se volají metody `setLowCutoffFrequency` a `setHighCutoffFrequency`.

8.4 Nastavení efektu

Jak bylo uvedeno, standard VST dovoluje uložit a načíst nastavení parametrů pluginu při uložení projektu DAW aplikace. Třída `AudioProcessor` frameworku JUCE má pro tento účel metody. Metoda `getStateInformation` se volá při ukládání projektu v DAW. Jejím parametrem je reference na objekt typu `MemoryBlock`. Ten představuje blok paměti s volitelnou velikostí pro zápis dat. Opakem je metoda volaná při otevření projektu v DAW `setStateInformation`, která má dva parametry, a to ukazatel na paměť a velikost v bytech.

Aktuální stav pluginu se v projektu ukládá ke zvukové stopě, ke které je plugin přiřazený. Vzhledem k široké škále možných nastavení v pluginu byla navíc implementována možnost ukládat je do textového souboru. Uživatel pluginu tak má možnost použít jednu z přednastavených konfigurací nebo uložení vlastních. Díky tomu je možné sdílet různá nastavení mezi různými stopami, projekty a DAW aplikacemi.

Vlastní formát uložení má tvar textového řetězce podle obrázku 8.5. Každé nastavení má název ve tvaru `User <datum a čas>` ukončený středníkem následován parametry pásem, kde každé pásmo je ukončeno středníkem

a každý parametr je ukončen dvojtečkou. Parametry jsou horní zlomová frekvence, frekvence a rozsah oscilace kolem horní zlomové frekvence, stav aktivní/neaktivní, zesílení, prahová hodnota, výstupní úroveň a typ zkreslení. Stejný formát je použitý pro ukládání aktuálního stavu pluginu.

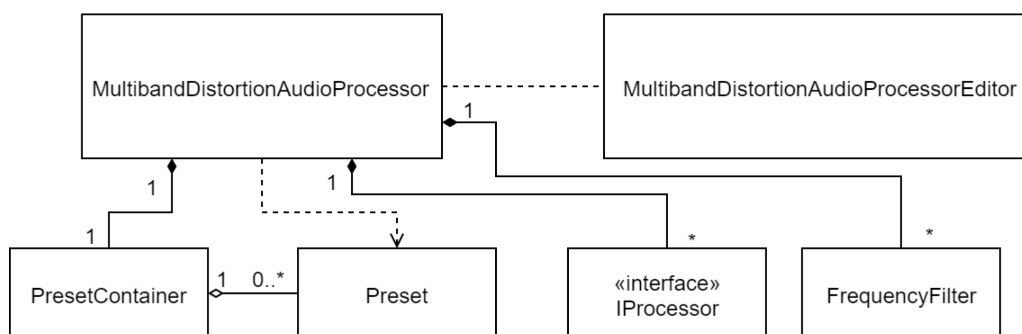
```
nazev_1;param_1_pasmo_1:param_2_pasmo_1:;param_1_pasmo_2:
param_2_pasmo_2:;|nazev_2;param_1_pasmo_1:param_2_pasmo_1
:;param_1_pasmo_2:param_2_pasmo_2:;|
```

Obrázek 8.5: Formát uložení nastavení

Po načtení jsou nastavení v paměti a k dispozici pro výběr v uživatelském rozhraní. Pro nastavení je určená třída `Preset`, která se stará o serializaci, deserializaci a poskytnutí dat při aplikaci nastavení. Všechna nastavení jsou uložena jako instance třídy `PresetContainer`, která se stará o přidávání a odebírání uživatelských nastavení, serializaci a deserializaci.

8.5 Procesor

Třída `MultibandDistortionAudioProcessor` je potomkem třídy `AudioProcessor` z frameworku JUCE. Základ implementace této třídy byl při založení projektu automaticky generován programem Projucer. Tato třída je jádrem efektu, protože provádí zpracování zvukového signálu v metodě `processBlock`, nastavení parametrů přenosu v metodě `prepareToPlay`, dále se stará o ukládání a načítání stavu, nastavení parametrů efektu a vytvoření instance třídy grafického editoru pro ovládání pluginu `MultibandDistortionAudioProcessorEditor`.



Obrázek 8.6: Diagram tříd s vazbami na procesor

Na obrázku 8.6 jsou znázorněny závislosti tříd popsaných v předchozích

sekcích. Jak bylo uvedeno, každé frekvenční pásmo tvoří jedna struktura `ProcessingBand` s instancí filtru a instancí třídy pro zkreslení. Ty jsou uloženy ve vektoru `processingBands`. Třída procesoru má veřejné metody pro čtení a nastavení všech parametrů podle indexu pásma nebo podle indexu zlomové frekvence mezi pásmy. Pro přidání dalšího pásma se volá metoda `addProcessingBand`, která rozšíří vektor pásem a vektor zlomových frekvencí `frequencyCutoffs` o další záznamy. Pokud se přidává další pásmo za běhu pluginu poprvé, přidá se zkreslení typu *Distortion* s parametry $gain = 0$ dB, $threshold = 1$ a $level = 0$ dB. Zlomová frekvence mezi posledním a novým pásmem se nastaví jako hodnota mezi poslední zlomovou frekvencí a nejvyšší možnou frekvencí 20 kHz. Třída má pro informaci, kolik je aktivních pásem, atribut s počtem aktivních pásem `activeProcessorsCount`, který se při přidání nového pásma inkrementuje. Při zpracování v metodě `processBlock` se vzorek zpracuje pouze počtem pásem daným touto proměnnou. Při odebrání pásma se volá metoda `removeProcessingBand`, která neodstraňuje záznamy ve vektorech, ale pouze sníží počet aktivních pásem o jedna. Při opětovném přidání pásma se použije předchozí nastavení.

Protože změna počtu pásem je provedena z vlákna pro uživatelské rozhraní (UI vlákno), může při ní docházet k souběhu a pádu pluginu, protože zpracování zvukových dat probíhá ve vlastním vlákně. Framework JUCE poskytuje zámek, který vrací metoda `getCallbackLock` jako instanci třídy `CriticalSection`. Nad ní je možné zavolat metody `enter` a `exit` pro vstup a opuštění kritické sekce, která je automaticky uzamčena, zatímco hostitelská aplikace volá metodu `processBlock`. Tento zámek se používá pro volání z UI vlákna pro uzamčení přístupu k proměnným, které se používají při zpracování zvuku [6]. Uzamčení by nemělo trvat dlouho nebo být voláno často, protože se to může projevit na výstupu zvuku. V tomto případě zámek problémy nezpůsobuje, protože se používá jen při změně počtu aktivních pásem nebo změně typu zkreslení, kdy se například odstraní z paměti instance původního zkreslení a nahrazuje se novou. Tyto události jsou vyvolané uživatelem a nedochází k zamykání příliš často ani na delší dobu.

8.6 Editor

Implementace grafického editoru je ve třídě `MultibandDistortionAudioProcessorEditor`, jejíž základ je také generován frameworkem JUCE. Třída vytváří okno editoru s ovládacími prvky (obrázek 8.7). Instance této třídy je vytvořena v procesoru v metodě `createEditor` a v konstruktoru předává ukazatel na vlastní instanci, která je v editoru uložená jako atribut pro volání

změn a čtení hodnot parametrů ovládacích prvků.

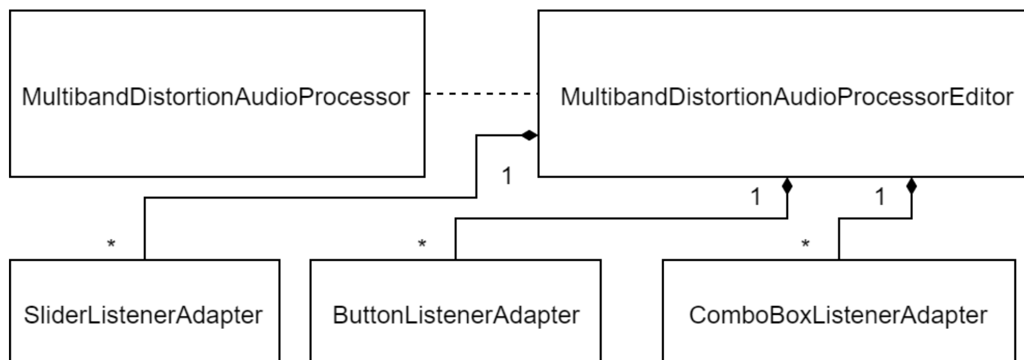
Ovládacím prvkům v grafickém rozhraní JUCE se pro naslouchání události nastavuje ukazatel na objekt posluchače. Obvykle roli posluchače zajišťuje třída editoru tak, že dědí abstraktní třídy posluchačů (např. `SliderListener` pro posuvníky, `ComboBoxListener` pro výběrové seznamy apod.) a přepisuje metody pro zpracování události. Tato metoda může obsluhovat všechny ovládací prvky stejného typu, jako například posuvníky v metodě `sliderValueChanged`, které se předává hodnota a zjišťuje se, který konkrétní prvek událost vyvolal. Na základě toho se volá změna parametru nad instancí procesoru.

Obrázek 8.7: Grafické rozhraní pluginu

Ovládací prvky v pluginu by se daly rozdělit do dvou kategorií na prvky, které se změnou počtu pásem nemění, a na ty, jejichž počet se mění. První jsou tlačítka pro ukládání a načítání nastavení a posuvník pro změnu počtu pásem. Události z nich jsou zpracovány tak, jak je popsáno výše. Ovládací prvky, které souvisí se zpracováním zvuku v jednotlivých frekvenčních pásmech se dynamicky přidávají a odebírají. Prvky nastavení zkreslení jsou sdruženy ve struktuře `BandControls` a ukládají se ve vektoru velikosti

rovné počtu pásem podobně jako prvky pro nastavení zlomových frekvencí ve struktuře `CutoffControls`.

Popsaná reakce na událost by byla komplikovaná, protože by v metodě jednoho posluchače bylo potřeba zjistit, který prvek ze kterého pásma událost vyvolal, a na základě toho volat metodu procesoru pro změnu parametru podle indexu pásma nebo zlomové frekvence. V tomto případě by bylo vhodné nastavit prvku zpracování události pomocí lambda funkce. Takovou možnost ovládací prvky v JUCE nemají a posluchač se nastavuje jako ukazatel na objekt. Z toho důvodu byly vytvořeny vlastní třídy posluchačů, které jsou znázorněny na obrázku 8.8. Stejně jako způsob použití třídy editoru jako posluchače každá třída (např. `SliderListenerAdapter` pro události posuvníků) dědí konkrétní třídu posluchače (např. `SliderListener`) a přepisuje metodu pro obsluhu události (např. `sliderValueChanged`), která volá funkci předávanou jako parametr v konstruktoru.



Obrázek 8.8: Diagram tříd s vazbami na editor

9 Testování

Testování stability a funkčnosti, kterému předcházelo měření zpoždění pluginu, proběhlo v pěti DAW aplikacích. Protože se jedná o efekt zkreslení určený primárně pro zkreslení elektrických kytar, byl plugin testován se třemi kytaristy, kteří poskytli zpětnou vazbu na kvalitu zvuku a ovládání.

Pro nahrávání byl použit jako zvuková karta kytarový multiefekt Zoom G2.1 Nu. Testování proběhlo na počítači s následujícím vybavením.

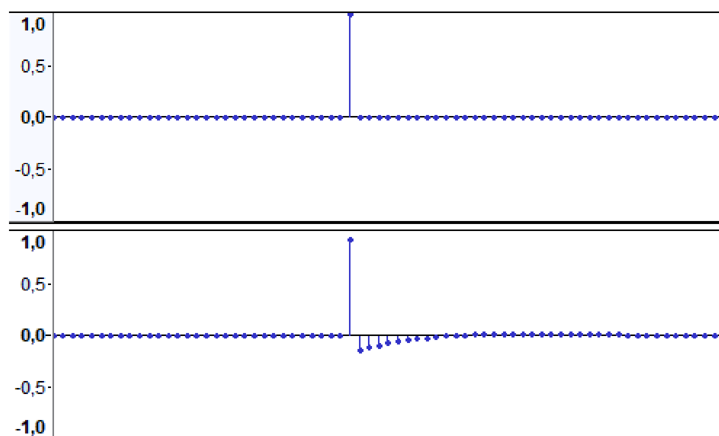
- Procesor Intel Core i7-7500U CPU
  - 2.70GHz
  - jádra: 2 (logické procesory: 4)
- 8GB DDR4 RAM
- Windows 10

9.1 Měření zpoždění

VST pluginy mohou mít nežádoucí zpoždění. To je způsobeno například použitím některých filtrů, které mohou vstupní signál zpožďovat, a může tak dojít k problémům při smíchání nahrávky v případě, že projekt obsahuje více stop. Příkladem může být to, že kytara bude oproti ostatním nástrojům o několik vzorků zpožděná. V případě, že jde o malé nebo žádné zpoždění, výsledné smíchání je v pořádku. Velké zpoždění umí většina DAW kompenzovat. Jedním ze způsobů je ruční nastavení, kdy uživatel na základě znalosti zpoždění od výrobce pluginu zadá počet vzorků sám.

VST umožňuje pluginům reportovat známé zpoždění. Na jeho základě může také hostitelská aplikace kompenzovat zpoždění automaticky. Framework JUCE tuto funkcionalitu podporuje implementací metody `getLatencySamples` třídy procesoru `AudioProcessor` [6].

Algoritmy zkreslení v navrženém efektu nezpůsobují žádné zpoždění. Použitý filtr z JUCE by zpoždění produkovat mohl. V dokumentaci filtru se jeho zpoždění neuvádí, a tak byl proveden test, při kterém se pluginu nechal zpracovat jednotkový impuls, který se následně porovnal s výstupem. Výsledek měření je na obrázku 9.1, kde je v horní polovině jednotkový impuls a ve spodní výstup pluginu s krátkou odezvou. Při měření byl nastaven efekt na tři pásma se zlomovými frekvencemi 200 Hz a 800 Hz, z nichž každé mělo



Obrázek 9.1: Odezva efektu na jednotkový impuls

zkreslení typu *Distortion* s hodnotou zesílení a výstupu 0 dB a prahovou hodnotu rovnou jedné, aby nedocházelo ke zkreslení.

Na základě měření nebyla implementována metoda `getLatencySamples` a plugin tak reportuje nulové zpoždění. Pro ověření, že plugin nezpůsobuje nežádoucí zpoždění při nahrávání a smíchání stop, byla vytvořena krátká nahrávka kytary společně s bicími, která je součástí této práce.

9.2 Testování v hostitelských aplikacích

Pro testování funkčnosti a stability byly vybrány DAW aplikace Reaper 5, FL Studio 12, Cubase LE AI Elements 9.5, Adobe Audition CC 2018 a Tracktion 6. Kromě poslední jmenované bylo testování provedeno ve zkušebních verzích.

Ve všech aplikacích proběhlo testování podle stejného scénáře. Byly nahrány dvě zvukové stopy s jedním vstupním kanálem a vzorkovacími frekvencemi 44100 a 96000 Hz a dvě stopy se dvěma vstupními kanály o stejných vzorkovacích frekvencích. Všechny stopy obsahují čistý kytarový signál. Ty byly po jedné exportovány z aplikace s použitím vyvinutého pluginu s připraveným nastavením *Distortion*, které má tři kanály a v nich nastavena zkreslení *Overdrive*, *Distortion* a *LPCDistortion*. V každém DAW bylo dále otestováno uložení stavu při uložení projektu a ukládání a načítání uživatelských nastavení ze souboru.

Ve všech aplikacích proběhlo testování úspěšně. Podařilo se zavést plugin, aplikovat na připravenou zvukovou stopu a úspěšně exportovat do souboru. Také ukládání stavu pluginu v projektu a ukládání a načítání uživatelských nastavení bylo úspěšné. Původní testovací stopy i exportované zvukové stopy

ze všech DAW aplikací jsou součástí této práce.

9.3 Uživatelské testování

K testování přijatelnosti výsledného zvuku, ovládání a použití pluginu byli přizváni tři kytaristé, kteří mají vlastní zkušenosti s hudební technikou při vystupování i nahrávání v různých hudebních žánrech (metal, ska–punk). Se všemi probíhalo testování za stejných podmínek. Někteří zúčastnění plugin testovali poté i sami ve vlastním prostředí.

Plugin byl zaveden do aplikace Tracktion 6. Zvukový vstup se nenahrával ani neexportoval, ale upravený se posílal okamžitě na výstup. Tuto funkci mají DAW aplikace, aby se muzikant sám slyšel během nahrávání. Všichni tak mohli s připojenou kytarou hrát a ovládat plugin v reálném čase stejně jako fyzické zařízení. Po testování bylo uživatelům položeno několik otázek týkajících se hodnocení a názorů na kvalitu a přijatelnost výsledného zvuku, ovládání a použití pluginu. Všechny jejich odpovědi jsou zaznamenány v příloze B.

Hodnocení přijatelnosti zvuku a možností zkreslení je kladné. Všichni uvedli, že by si dovedli představit použití pluginu v praxi při nahrávání nebo cvičení. Uvedli i několik nedostatků a možných vylepšení, které se týkají ovládání, ukládání nastavení a celkového nastavení zvuku, které mohou být užitečné pro další vývoj pluginu.

10 Závěr

Cílem práce bylo seznámení se s technologií VST a návrh a implementace digitálního zvukového efektu jako VST pluginu pro použití v hostitelských aplikacích DAW pro nahrávání a zpracování zvukového signálu.

Před vlastním návrhem byly prostudovány a popsány základní techniky reprezentace a zpracování digitálního zvukového signálu, několik dostupných open-source implementací zvukových efektů realizovaných jako VST pluginy a dostupné nástroje pro vývoj VST pluginů a zvukových aplikací. Z prozkoumaných vývojových nástrojů byl pro implementaci vlastního pluginu vybrán framework JUCE a součástí práce je i sada několika jednoduchých efektů realizovaných s jeho pomocí.

Výsledkem této práce je implementace navrženého efektu vícepásmového zkreslení, kde se vstupní signál filtrací rozděluje do několika frekvenčních pásem, kterým lze nastavit různé algoritmy a parametry zkreslení zvlášť, a umožnit tak více možností zkreslení výsledného zvuku. Kromě běžných zkreslovacích algoritmů a funkcí, které jsou v pluginu implementovány, byl navržen a implementován i vlastní algoritmus zkreslení založený na lineární predikci a odhadu budoucích vzorků.

Plugin byl úspěšně testován v pěti různých aplikacích DAW pro ověření správné funkce a stability. Protože efekt zkreslení má využití hlavně při hře na elektrickou kytaru, byl poslechový test přijatelnosti výsledného zvuku a test ovládání pluginu proveden se třemi kytaristy, kteří mají vlastní zkušenosti s hudební technikou. Z výsledků testování vyplývá, že plugin je stabilní a plnohodnotný efekt, který lze použít pro zkreslení kytarových stop při nahrávání ve studiu nebo pro domácí použití jako náhrada fyzických efektů nebo zesilovačů.

Literatura

[1] Anderton, C. *Multiband Processing: The Next Big Thing in Effects?* [online]. 2012. [cit. 2018/03/25]. Dostupné z: https://www.sweetwater.com/insync/multiband-processing-technique-effects/.

[2] Collomb, C. *Linear Prediction and Levinson-Durbin Algorithm* [online]. 2009. [cit. 2018/03/24]. Dostupné z: http://www.emptyloop.com/technotes/A%20tutorial%20on%20linear%20prediction%20and%20Levinson-Durbin.pdf.

[3] De Man, B. - Reiss, J. D. Adaptive control of amplitude distortion effects. *Audio Engineering Society Conference: 53rd International Conference: Semantic Audio*. 2014. Dostupné z: http://www.aes.org/e-lib/browse.cfm?elib=17118.

[4] Černocký, J. *Zpracování řečových signálů* [online]. 2006. [cit. 2018/03/11]. Dostupné z: http://www.fit.vutbr.cz/study/courses/ZRE/public.

[5] *dsp::StateVariableFilter::Filter Class Template Reference* [online]. ROLI. [cit. 2018/04/11]. Dostupné z: https://docs.juce.com/master/classdsp_1_1StateVariableFilter_1_1Filter.html.

[6] *AudioProcessor Class Reference* [online]. ROLI. [cit. 2018/04/22]. Dostupné z: https://docs.juce.com/master/classAudioProcessor.html.

[7] Roelandts, T. *How to Create a Simple Low-Pass Filter* [online]. [cit. 2018/03/13]. Dostupné z: https://tomroelandts.com/articles/how-to-create-a-simple-low-pass-filter.

[8] Roelandts, T. *Impulse Response* [online]. [cit. 2018/04/19]. Dostupné z: https://tomroelandts.com/articles/impulse-response.

[9] Schaedler, J. *Seeing circles, sines and signals* [online]. [cit. 2018/03/01]. Dostupné z: https://jackschaedler.github.io.

[10] Smith, J. O. *Introduction to digital filters* [online]. W3K Publishing, 2007. [cit. 2018/03/11]. Dostupné z: https://www.dsprelated.com/freebooks/filters.

[11] Smith, J. O. *Mathematics of the Discrete Fourier Transform (DFT)* [online]. W3K Publishing, 2007. [cit. 2018/02/04]. Dostupné z: https://ccrma.stanford.edu/~jos/mdft.

[12] Smith, J. O. *Physical signal audio processing* [online]. W3K Publishing, 2010. [cit. 2018/03/07]. Dostupné z: `https://www.dsprelated.com/freebooks/pasp`.

[13] Smith, J. O. *Spectral Audio Signal Processing* [online]. W3K Publishing, 2011. [cit. 2018/04/19]. Dostupné z: `https://www.dsprelated.com/freebooks/sasp`.

[14] Smith, S. W. *The Scientist and Engineer's Guide to Digital Signal Processing* [online]. 1997. [cit. 2018/02/11]. Dostupné z: `http://www.dspguide.com`.

[15] VST 3 API Documentation. *VST 3 SDK for developing VST Plug-in.* Steinberg Media Technologies GmbH, 2017.

[16] Sweetwater. *Low Frequency Oscillator (LFO)* [online]. 1997. [cit. 2018/03/07]. Dostupné z: `https://www.sweetwater.com/insync/low-frequency-oscillator-lfo`.

A Obsah přiloženého DVD

Součástí práce jsou následující přílohy, které jsou na přiloženém DVD:

- přeložené VST pluginy pro 64 bitové verze Windows ve složce `bin`,
- zdrojové kódy pluginů včetně souborů pro otevření v programu Projucer a vygenerovaných projektů pro Microsoft Visual Studio ve složce `src`,
- elektronická verze textu diplomové práce se zdrojovými soubory ve složce `doc`,
- nahrávky z testování pluginu v DAW aplikacích ve složce `recording`,
- poster ve složce `Poster`.

B Uživatelské hodnocení

Kvalita výsledného zvuku

- V Cubase 7 jsem si všiml, že mi signál kolísá, když plugin spustím, ale když silně zahraji na kytaru, tak to signál oživí. Při zvyšování levelu na libovolném pásmu vzniká nepříjemné praskání – příliš přebuzená hlasitost, kterou reproduktory nezvládají. Celkově to při vysokém gainu vůbec nešumí – to je za mě velké plus.
- Zvuk při výstupu je hezky čitelný. Lepší samozřejmě při hlasitějším hraní.
- Kvalitní výstup – plně se vyrovná klasickým krabičkovým efektům. Vyzkoušeno na reproboxu i na repro sadě k PC. Zvuk lze samozřejmě přebudit a nastavit příliš silný signál, ale při standartním použití je zvuk hezký a „čistý“ bez vnějších vlivů (šum, praskání).

Kvalita a rozsah zkreslení (např. v porovnání s existujícími efekty)

- Zkreslení je pěkné a čisté. Je potřeba si nakroutit to svoje a dokázal bych si představit, že bych to využil při nahrávání demo nahrávek a nebo jen pro cvičení. Zkreslení mi přijde „Marshallovské“ od jemného overdrive až po silný hi-gain.
- Zkreslení je pěkně čitelné. Líbí se mi, že se dá nastavit od jemnějšího, lehce nakřáplého zvuku, až k silnému „metalovému“ zkreslení. Dobře nastavitelný overdrive zvuk. Možná bych přidal ještě nějakou možnost modifikace zvuku.
- Vysoký rozsah škálovatelnosti dovoluje vytvořit si velice specifický zvuk. Na všech úrovních byl však zvuk čistý a čitelný. Přednastavené zkreslení se chová podle očekávání a zvuk bez abnormalit. Použitelné i pro hráče nehrající pouze metal ale i jemnější styly (ska/punk. . .)

Možnost nastavení frekvenčních pásem s volbou zkreslení

- Přijde mi až zbytečné ovládat třeba basové pásmo, kdy při zvyšování gainu jen zvyšuji hlasitost, zvuk pak zní moc jako fuzz a celkově to pak zakrývá ostatní pásma. To samé platí u vysokých frekvencí, které jsou už neslyšitelné. Oproti reálnému aparátu je toho nastavení až příliš moc.

- Myslím, že nastavení frekvenčních pásem je až příliš. Něco jako klasický ekvalizér by mohl stačit.
- Pět pásem je pro mě zbytečně mnoho.

Funkci oscilace mezi přechody pásem

- Má to velký vliv na výsledný zvuk. Za mě je to velká výhoda. Úplně to mění charakter zkreslení.
- Nejsem si jistý jestli to můžu s jiným efektem porovnat. Výsledný zvuk je hladší.
- Nemám porovnání.

Vzhled uživatelského rozhraní

- Celkově přehledné, ale představil bych si vzhled podobný nějakému reálného zobrazení (ovladače na zesilovačích, přepínače atd.)
- Pěkně provedené, uživatelsky snadno ovladatelné i pro někoho kdo se tím nezabývá tolik do hloubky.
- Intuitivní, přehledné a vizuálně hezké.

Ovládání

- Ovládání mi přijde v pořádku.
- Nechal bych tak jak je, jak jsem psal výše, přehledné.
- Přidal bych potenciometru ruční vstup – zadáni hodnoty uživatelem.

Ukládání a načítání nastavení

- Ukládání a načítání presetu funguje v pořádku, jen při exportu se to někdy na pár vteřin zasekne.
- Nezkoušel jsem.
- Celkově uživatelsky příjemné, pouze bych navrhl přidat možnost „šablonu“ pojmenovat pro přehlednost.

Použití v praxi

- Tento VST plugin je za mě funkční a plnohodnotný pre-amp, který se dá využít na domácí nahrávání nebo cvičení. Dá se použít jako simulace zesilovače, ale ve výsledku aby byl zvuk použitelný, tak se

musí přidávat simulace kytarových impulsů (simulace reproboxu) což je u těchto pluginů běžná věc.

- Dovedu si představit použití ve studiu nebo na deformaci zvuku při domácím nahrávání.
- Komplexní využitelnost – díky možnostem škálování lze plugin využít pro experimentování ve zkušebně/doma, ale i jako plnohodnotný distortion ve studiu. Při zabudováni do el. efektu se nabízí i možnost použití při vystoupeních.

Chybějící funkce

- Představil bych si koncový ekvalizér, kompresor hlasitosti (limiter), celková úroveň gain.
- Mám rád fuzz, takže možná nějaká možnost přepnutí z overdrive na fuzz. Regulování koncové hlasitosti by také bylo v pohodě. Třeba by to ještě mohlo obsahovat ladičku.
- Žádnou další funkcionalitu bych nepřidával.