TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Aplikace pro zpracování audio signálu v reálném čase

Bakalářská práce

Studijní program: B0613A140005 Informační technologie
Studijní obor: Aplikovaná informatika

Autor práce: **Maxim Osolotkin**
Vedoucí práce: Ing. Miroslav Holada, Ph.D.
Ústav informačních technologií a elektroniky

Liberec 2022

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Aplikace pro zpracování audio signálu v reálném čase

Jméno a příjmení:	**Maxim Osolotkin**
Osobní číslo:	M19000032
Studijní program:	B0613A140005 Informační technologie
Specializace:	Aplikovaná informatika
Zadávající katedra:	Ústav informačních technologií a elektroniky
Akademický rok:	**2021/2022**

Zásady pro vypracování:

1. Seznamte se s problematikou digitálních audio efektů. Proveďte rešerši známých softwarů pracujících s audio efekty.
2. Navrhněte software pro PC, který bude v reálném čase zpracovávat audio signál. Implementujte v něm vybrané audio efekty (například ekvalizér, delay, chorus, vibrato, wah-wah, phaser a další nelineární efekty).
3. Navržený software realizujte. Ke konfiguraci digitálních audio efektů naprogramujte vlastní GUI.
4. Funkčnost realizovaného software ověřte, zdokumentujte a diskutujte jeho výhody a nevýhody.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30-40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] Zölzer U. et al.: DAFX: Digital Audio Effects, Wiley, ISBN-13: 978-0471490784,2002.
[2] Chassaing R.: Digital Signal Processing and Applications with the TMS320C6713 and TMS320C6416 DSK, Wiley-IEEE Press, ISBN-13: 978-0470138663, 2008.
[3] Davídek, V., Sovka, P.: Číslicové zpracování signálů a implementace, 1. vydání, Vydavatelství ČVUT , Praha 1996, 80-01-01530-0.

Vedoucí práce:	Ing. Miroslav Holada, Ph.D. Ústav informačních technologií a elektroniky
Datum zadání práce:	12. října 2021
Předpokládaný termín odevzdání:	16. května 2022

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

prof. Ing. Ondřej Novák, CSc.
vedoucí ústavu

V Liberci dne 19. října 2021

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

20. dubna 2022 Maxim Osolotkin

Aplikace pro zpracování audio signálu v reálném čase

Abstrakt

Cílem teto práce je návrh a implementace programu pro zpracovaní audio signálu v reálném čase pro operační systém Windows v jazyce C++. Zpracování audia je řešeno za pomoci pluginů, které jsou ve formě DLL. Častí práce je tedy i návrh rozhraní pro komunikaci mezi programem a pluginy se zaměřením na jednoduchost v jazyce C. Rozhraní je následně využito k implementaci vybraných audio efektů. Konkrétní implementace jsou popsané a relevantní efekty otestované metodou black-box k ověření jak funkčnosti jednotlivých pluginů, tak i programu samotného. Práce také obsahuje přehled již existujících real-time audio aplikací, audio pluginů a audio protokolů ke komunikaci se zvukovou kartou.

Klíčová slova: zpracovaní audio signálu v reálném čase, audio efekt, plugin

Real-time audio signal processing application

Abstract

The aim of this thesis is to design and implement a program for real-time audio signal processing under Windows operating system in C++ language. The audio processing is handled with plugins which are in the form of DLL. Thus, part of the work is the design of the interface for communication between the program and the plugins, focusing on simplicity in C. The interface is then used to implement selected audio effects. The concrete implementations are described and the relevant effects are tested using the black-box method to verify the functionality of both the plugins and the program itself. The work also includes a survey of existing real-time audio applications, audio plugins and audio protocol to communicate with the sound card.

Keywords: real-time audio signal processing, audio effect, plugin

Poděkování

Nechť je mou osobou děkováno panu Ing. Miroslavu Holadovi, Ph.D za snahu o zlepšení těchto listů a veškerou pomoc během práce.

Obsah

Seznam zkratek 12

Úvod **13**

1 Úvod do teorie a základní pojmy **14**
1.1 Digitalní signál 14
1.2 Real-time zpracování 15
1.3 Diskrétní Fourierova transformace 15
1.3.1 Rychlá Fourierova transformace 17
1.3.2 Inverzní Diskrétní Furierova transformace 17
1.4 Filtry 17
1.4.1 Filtr s konečnou impulzní odezvou 18
1.4.2 Filtr s nekonečnou impulzní odezvou 18
1.4.3 Pásmová propust 19
1.5 Okénkovací funkce 20
1.6 Modulace 22

2 Přehled Real-time audio aplikací **23**
2.1 Digital audio workstation software 23
2.1.1 Audacity 23
2.1.2 REAPER 23
2.1.3 Cubase 24
2.1.4 Pro Tools 24
2.1.5 GarageBand 24
2.2 Audio pluginy 24
2.2.1 Virtual Studio Technology 25
2.2.2 Audio Units 25
2.2.3 Avid Audio eXtension 25
2.2.4 Guitar Rig 25
2.3 Audio komunikační protokoly 25
2.3.1 Windows Audio Session API 25
2.3.2 Steinberg Audio Streaming Input Output 26
2.3.3 ASIO4ALL 26

3 Praktický vývoj aplikace **27**
3.1 Uživatelské rozhráni 28

3.2 Obsluha audia 29
3.2.1 ASIO rozhrání 29
3.2.2 WASAPI rozhrání 30
3.2.3 Legacy rozhrání 30
3.2.4 Souborový vystup a výstup 30
3.3 Předvolby 31
3.4 Návrh rozhrání pluginu 32
3.5 Výsledný stav 36

4 Implementované audio efekty **37**
4.1 Zesílení 37
4.2 Zkreslení 38
4.3 Delay 39
4.4 Phaser 40
4.5 Vibrato 41
4.6 Chorus 42
4.7 Octaver 43
4.8 Reverb 44
4.9 Biquad Filter 45

5 Závěr **47**

Seznam literatury **49**

A Příloha **52**

Seznam obrázků

Obrázek 1.1:Pásmové propusti . . . 19
Obrázek 1.2:Hannovo okno . . . 21
Obrázek 1.3:Vliv okénkovací funkce na frekvenci . . . 21
Obrázek 1.4:Modulace amplitudy . . . 22

Obrázek 3.1:Schéma návrhu programu . . . 27
Obrázek 3.2:Plugin a program . . . 34
Obrázek 3.3:Uživatelské rozhraní . . . 36

Obrázek 4.1:Zesílení o 4 dB . . . 37
Obrázek 4.2:Ořezávací funkce . . . 39
Obrázek 4.3:Delay, odezva jednotkového impulzu . . . 40
Obrázek 4.4:Phaser, odezva bílého šumu . . . 41
Obrázek 4.5:Vibráto, spektrogram výstupu sinusovky 100 Hz . . . 42
Obrázek 4.6:Octaver, frekvenční spektrum výstupu pro vstup o 220 Hz . . . 44
Obrázek 4.7:Reverb . . . 45
Obrázek 4.8:Biquad filter, odezva na bílý šum . . . 46

Seznam zdrojových kódu

3.1 Vytvoření uživatelského rozhraní pluginu s jednou kontrolkou 33
4.1 Phase vocoder, výpočet phase s ohledem na korekci 43
4.2 Vypočet koeficientu low pass filtru 46

Seznam zkratek

TUL	Technická univerzita v Liberci
DFT	Discrete Fourier transform
FFT	Fast Fourier transform
IDFT	Inverse discrete Fourier transform
IFFT	Inverse fast Fourier transform
FIR	Finite impulse response
IIR	Infinite impulse response
MATLAB	Matrix laboratory
API	Application Programming Interface
winAPI	Windows Application Programming Interface
DAW	Digital audio workstation
IDE	Integrated development environment
DLL	Dynamic-link library
MIDI	Musical instrument digital interface
SDK	Software development Kit
OS	Operating system

Úvod

Se zpracováním audio signálu se lze setkat zcela běžně, např. při komunikaci skrz mobilní telefon, nebo při přehrávání hudby. V takovýchto případech samotné zpracování audia ale není zdaleka přímým cílem. S cíleným zpracováním audio signálu se lze setkat např. v hudebním průmyslu, kde zpracování zvuku je nezbytnou součástí jak nahrávaní, tak i živého hraní.

V digitálním světě se standardem staly programy zvané Digital Audio Workstation (DAW) v kombinaci s různými audio pluginy. Tyto programy nabízí velkou flexibilitu jak pro nahrávaní, tak i real-time zpracování. Většina kvalitních programů, nebo i samotných pluginů, je však placená, a ne vždy se lze setkat s přijatelnou cenou, jelikož se taky jedná o software používaný profesionály v hudebním průmyslu. V dnešní době je již dostatek výkonu, aby podobné programy mohly být zprovozněné i na běžně dostupném hardwaru (např. notebook, mobilní telefon). Takovéto programy ale nemusí být vždy ideální a někdy se může hodit jednodušší program zaměřující se jen na jednu, nebo úzké spektrum funkcí.

Vznikl podnět k implementaci jednoduchého programu, který by pomocí pluginů řešil zpracování audio signálu v reálném čase. Program byl psán pro operační systém Microsoft Windows v jazyce C++ a pluginy byly představeny jako DLL soubory, jejichž komunikaci s programem zprostředkovává navržené rozhraní v jazyce C. Při návrhu a implementaci programu se snažilo docílit minimální závislosti na operačním systému, aby program byl případně jednoduše přenositelný na jiné operační systémy, či pouze procesory. Při návrhu rozhraní se převážně dbalo na jednoduchost, aby se především při implementaci pluginů dalo soustředit na samotné zpracování signálu. Tento program lze využít jako např. kytarový procesor, který by umožňoval zaujatějším jedincům si relativně jednoduchým způsobem implementovat vlastní audio efekty.

1 Úvod do teorie a základní pojmy

Na začátek je vhodné uvést alespoň některé pojmy, se kterými se dá setkat v následujícím textu, když už ne kvůli definicím samotným, tak alespoň kvůli stanovení jakési jednotnosti textu a zmenšení jeho míry abstrakce.

1.1 Digitalní signál

V digitálním světě nelze nijak vyjádřit v paměti reálný spojitý signál, jelikož mezi jeho dvěma libovolnými body vždy bude alespoň jeden bod další. Není tedy možné uchovat všechny body v konečné paměti. Proto se spojitý signál reprezentuje za pomoci signálů digitálních (číslicových). [1, 2]

Digitální signál je představen konečným počtem hodnot spojitého signálu v určitých časech. Proces převodu spojitého signálu na konečný počet diskrétních hodnot, neboli vzorků, se nazývá vzorkování. Pokud krok mezi časy je při vzorkování konstantní, k popisu se používá tzv. vzorkovací frekvence, která udává počet převedených vzorku za sekundu.

Poznámka 1.1. Vztah mezi vzorkovací frekvenci f_s a časovým intervalem T v sekundách mezi dvěma následujícími vzorky můžeme definovat jako

$$T = 1/f_s$$

Je zřejmé, že se při vzorkování ztrácí data. Tyto ztráty lze snížit zvětšením vzorkovací frekvence. Ovšem, zvýšení nese i větší nároky na zpracování, tedy je vhodné vzorkovací frekvenci volit relativně k snímaným datům. K tomu může být nápomocný Nyquistův vzorkovací teorém, viz tvrzení 1.2, který pojednává o vztahu vzorkovací frekvence a maximální zachycené frekvence při snímaní.

Tvrzení 1.2. Nechť signál vzorkujeme frekvenci f_s, pak maximální frekvence f_{max}, kterou můžeme zachytit je daná vztahem [3]

$$f_{max} = f_s/2$$

Ovšem, reálný spojitý signál má nejen nekonečný počet bodů, ale i jeho hodnoty nemusí tvořit konečnou množinu. Proces převodu hodnot z jednoho rozsahu na

konečný počet hodnot rozsahu druhého se nazývá kvantováni. Kvantovat lze buď s konstantním krokem, což je standardní cesta pro záznam audia, nebo proměnlivým, což třeba umožňuje použit více hodnot pro vybrané pásmo.

Digitální signál je tedy vyjádření spojitého signálu o konečném počtu vzorků a konečném počtu jejich hodnot.

1.2 Real-time zpracování

Real-time zpracování signálu se může charakterizovat jako postupné zpracovaní signálu, při kterém se odpovídající vzorek vypočítá ze současného a již zpracovaných vzorků.

Tento postup je vhodný pro zpracování velkých dat, která se nedají uchovat v paměti současně. Nebo, typičtěji, data, která se průběžně získávají a nejsou dostupná hned, ale je na ně potřeba okamžitě reakce.

Důležitým atributem real-time zpracovaní je latence, která vyjadřuje časové zpoždění mezi vstupem a výstupem vzorku. Formálně by mohla být definovaná následovně.

Definice 1.3. Nechť t_1 je čas, kdy libovolný vzorek x diskrétního signálu X byl předán k real-time zpracování. Nechť t_2 je čas kdy vzorek x byl zpracován. Pak latenci real-time zpracování δ vzorku x vyjádříme jako $\delta = t_2 - t_1$.

Zpracování signálu obvykle neprobíhá přímo po jednotlivých vzorcích, ale po částech. Vzorky jsou tedy postupně načítané do bufferu a po jeho zaplnění se zpracovávají současně. Pro to, aby tohoto bylo možno docílit, je potřeba dvou bufferů. Data se načtou do prvního a začnou se zpracovávat. Mezitím jsou již nové vzorky zapisované do druhého. Následně, při střídaní buferrů, se tento proces opakuje. Tato metoda se označuje jako double buffering. Latenci takového systému lze vyjádřit jako dvojnásobek času potřebného k zaplnění jednoho bufferu.

Poznámka 1.4. Zpracováváme-li signál za pomoci double bufferingu se vzorkovací frekvencí f_s a délce bufferu N, pak latenci δ zpracování v sekundách vyjádříme následovně

$$\delta = \frac{2N}{f_s}$$

1.3 Diskrétní Fourierova transformace

K provedení frekvenční analýzy signálu lze využit diskrétní Fourierovy transformace, zkráceně DFT.[4]

Definice 1.5. Nechť $\forall x_n \in C$, kde $n \in Z, n \in \langle 0, N)$, pak definujeme DFT následovně

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk}, \quad k \in Z, k \in \langle 0, N)$$

Výsledkem DFT je komplexní vektor aproximačně definující původní signál jako množinu kosinusovek. Jejich součet v ideálním případě představuje původní signál. Každé komplexní číslo obsahuje informaci o fázi, amplitudě, která je rovna absolutní hodnotě daného čísla a frekvenci, která se dá získat z indexu a vzorkovací frekvence (viz. poznámka 1.6). Tím je každá kosinusovka jednoznačně určena.

Poznámka 1.6. Nechť X představuje výsledný komplexní vektor DFT komplexního signálu Y o délce N vzorkovaného frekvencí f_s, pak fázi p kosinusovky na indexu i můžeme vypočítat následovně

$$p_i = \arctan \frac{\mathrm{Re}(X_i)}{\mathrm{Im}(X_i)}$$

a frekvenci f_i příslušného indexu i vyjádříme jako

$$f_i = \frac{i \cdot f_s}{N}$$

Z poznámky 1.6 lze vidět, že počet hodnot kterými se kvantuje frekvence závisí na velikosti vstupního, respektive výstupního vektoru. K zvýšení rozlišení kvantování lze signál doplnit nulovými vzorky na potřebnou délku (tzv. zero-padding).

Jelikož maximální frekvence, kterou lze zachytit je $f_s/2$ (viz teorém 1.2) a výsledek pokrývá frekvenční spektrum po celé f_s, druhá polovina výsledného vektoru je vlastně redundantní. Jedná se o *symetrický komplexně sdružený* vektor, viz tvrzení 1.8, odvození lze vidět v poznámce 1.7.

Poznámka 1.7. Položíme-li

$$\{a_x\}_{x=1}^{\lceil N/2 \rceil - 1} = e^{-\frac{2\pi i}{N} nx}$$
$$\{a_y\}_{y=1}^{\lceil N/2 \rceil - 1} = e^{-\frac{2\pi i}{N} n(N-y)}$$

pak

$$a_y = \cos(-\frac{2\pi}{N} n(N-y)) + i \sin(-\frac{2\pi}{N} n(N-y)) =$$
$$= \cos(\frac{2\pi}{N} ny - 2\pi n) + i \sin(\frac{2\pi}{N} ny - 2\pi n) = \cos(\frac{2\pi}{N} ny) + i \sin(\frac{2\pi}{N} ny) =$$
$$= \cos(-\frac{2\pi}{N} ny) - i \sin(-\frac{2\pi}{N} ny) = \overline{a_x}$$

tedy můžeme vidět, že DFT reálného signálu je *komplexně sdruženě symetrické* od druhého prvku.

Tvrzení 1.8. Nechť X je DFT reálného signálu, pak pro $k \in \{1, 2, ..., \lceil N/2 \rceil - 1\}$ platí

$$X_k = \overline{X_{N-k}}$$

1.3.1 Rychlá Fourierova transformace

Jelikož přímá implementace DFT je náročná na výpočet, tak se v praxi používá rychlá Fourierova transformace.[5]

Rychlá Fourierova transformace, častěji známa pod zkratkou FFT (Fast Fourier transform), je jakýkoli algoritmus, který je schopen výpočtu diskrétní fourierovy transformace za amortizované časové složitosti $O(n \log n)$.

1.3.2 Inverzní Diskrétní Furierova transformace

K diskrétní Furierové transformaci existuje inverzní transformace (zkráceně značená jako IDFT), a, samozřejmě, i její rychlá varianta (IFFT).[4]

Definice 1.9. Nechť $\forall x_n \in C$, kde $n \in Z, n \in \langle 0, N)$, pak definujeme IDFT následovně

$$X_k = \frac{1}{N} \sum_{n=0}^{N-1} x_n e^{\frac{2\pi i}{N} nk}, \quad k \in Z, k \in \langle 0, N)$$

1.4 Filtry

Filtr je jakýsi modul, či systém, který modifikuje vstupní signál. Filtry se využívají k odstranění nechtěných příznaků signálu. V audio světě se většinou využívají k ovlivnění frekvenčního spektra signálu.[6]

K popisu chování filtru se dá použit tzv. přenosová funkce 1.10.

Definice 1.10. Uvážíme li časově neproměnný diskrétní filtr, pak můžeme přenosovou funkci $H(z)$ definovat jako

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

kde Y(z) představuje z-transformaci (1.11) výstupního y_n a X(z) z-transformaci vstupního x_n signálu filtru.[7]

Definice 1.11. Nechť z je libovolné komplexní číslo, $n \in Z$ a $\forall x_n \in \mathbf{T}$, kde $\mathbf{T}$ je těleso, pak z-transformaci X(z) můžeme definovat jako[8]

$$H(z) = \sum_{n=-\infty}^{\infty} x_n z^{-n}$$

Digitální filtry lze členit dle závislosti na impulzní odezvě, a to konečné (FIR), nebo nekonečné (IIR). Nebo také dle ovlivnění frekvenčního spektra - horní propust, dolní propust, atd.

Poznámka 1.12. Filtr je označován jako kauzální, pokud výstupní signál $y(n)$ je nulový pro všechna $n < 0$ pro daný vstupní signál $x(n) = 0$. Tedy filtr není schopen reakce na vstup, který se do něj ještě nedostal.[6]

1.4.1 Filtr s konečnou impulzní odezvou

FIR (finite impulse response), filtr s konečnou impulzní odezvou. Tedy takový filtr, jehož odezva se v konečném čase uchýlí k nule, jelikož je závislá jen na současném a minulých vzorcích.[6]

Popis vztahu vstupu a výstupu filtru lze vidět v následující definici.

Definice 1.13. Nechť $\mathbf{T}$ je těleso a $\forall x_n \in \mathbf{T}$, kde $n \in Z, n \in \langle 0, N)$ pak vztah vstupu x_n a vstupu y_n FIR filtru řadu N můžeme popsat následovně

$$y_n = \sum_{i=0}^{N} b_i x_{n-i}, \quad b_i \in T$$

A přenosovou funkci za podmínky kauzality (viz. 1.12) lze definovat následovně.

Definice 1.14. Nechť z je libovolné komplexní číslo, $k \in Z$ a $\forall b_k \in \mathbf{T}$, kde $\mathbf{T}$ je těleso, pak přenosovou funkci $H(z)$ kauzálního FIR filtru řadu N můžeme definovat následovně

$$H(z) = \sum_{k=0}^{N-1} b_k z^{-k}$$

1.4.2 Filtr s nekonečnou impulzní odezvou

Filtr s nekonečnou impulzní odezvou, zkráceně IIR (infinite impulse response), je takový filtr, jehož odezva se v konečném čase neuchýlí k nule a bude trvat do nekonečna. Je to způsobeno tím, že kromě nynějšího a minulých vzorků odezva závisí i na předchozích odezvách.[6]

Popis vztahu vstupu a výstupu filtru je popsán v definici 1.15.

Definice 1.15. Nechť x_n je vstup IIR filtru řadu P pro vstupní sekvencí a Q pro výstupní, přičemž platí $\forall x_n \in \mathbf{T}$, kde $\mathbf{T}$ je těleso, pak výstup y_n bude definován následovně

$$y_n = \frac{1}{a_0}(\sum_{i=0}^{P} b_i x_{n-i} - \sum_{i=0}^{Q} a_j y_{n-j}), \quad b_i, a_j \in T$$

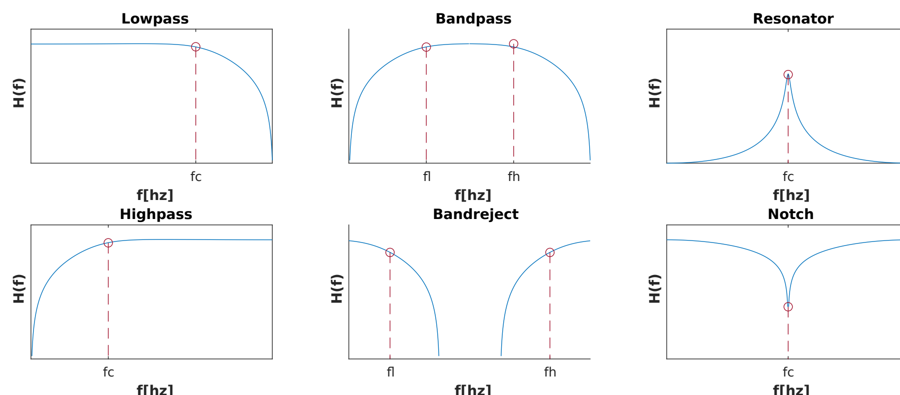
Přenosová funkce je definována následovně.

Definice 1.16. Nechť z je libovolné komplexní číslo, $\mathbf{T}$ je těleso a $\forall b_k, a_k \in \mathbf{T}$, kde $k \in Z$, pak přenosovou funkci $H(z)$ kauzálního IIR filtru řádu výstupu M a vstupu $N-1$, kde M a $N \in Z$ můžeme definovat následovně

$$H(z) = \frac{\sum_{k=0}^{N-1} b_k z^{-k}}{1 + \sum_{k=1}^{M} a_k z^{-k}}$$

1.4.3 Pásmová propust

Při zpracování audia se filtry převážně používají k ovlivnění frekvenčního spektra. Často se tedy lze setkat s rozdělením dle jejich vlivu na frekvenci signálu. Hlavním parametrem definujícím různé tzv. propustě je mezní (cutoff) frekvence, která označuje hranici počátku změny v propustnosti okolních frekvencí ve frekvenční odezvě filtru.[6]



Obrázek 1.1: Pásmové propusti

Dolní propust (Lowpass)

Filtr vybírá frekvence vyšší než mezní frekvence a tlumí nižší.

Horní propust (Highpass)

Filtr vybírá frekvence nižší než mezní frekvence a tlumí vyšší.

Pásmová propust (Bandpass)

Filtr vybírá frekvence mezi mezními frekvencemi f_{cl} a f_{ch} a zároveň tlumí frekvence nižší než f_{ch} a vyšší než f_{ch}.

Pásmová zádrž (Bandreject)

Filtr tlumí frekvence mezi mezními frekvencemi f_{cl} a f_{ch} a zároveň propouští frekvence nižší než f_{ch} a vyšší než f_{ch}.

Notch

Filtr tlumí frekvence jen v úzkém okolí mezní frekvence, ostatní frekvence se propouští.

Resonator

Filtr propouští frekvence jen v úzkém okolí mezní frekvence, ostatní frekvence jsou tlumené.

Všepropust (Allpass)

Filtr propouští všechny frekvence, avšak mění fázi signálu.

1.5 Okénkovací funkce

Pokud se signál zpracovává po částech, nad kterými se provádí operace, které vnímají onu část jako celek, např FFT, tak kvůli roztržení signálu se může jevit, že signál má jiné vlastnosti, např. periodu. [6]

Ke sníženi vlivu oříznuti signálu lze použit okénkovací funkci. Jedná se o funkci, která je nenulová na daném intervalu, jejíž hodnoty postupně určitým způsobem konverguji k nule na obou koncích. Tím se dociluje sníženi vlivu oříznutých okrajů, které, po pronásobení signálu hodnotami funkce, jsou méně ohodnocené než střední část signálu. Symetrická okénkovací funkce by mohla byt definovaná následovně.

Definice 1.17. Nechť $x, b \in R$ a $n \in N$ pak symetrickou okénkovací funkci $A(x)$ definujeme následovně [9]

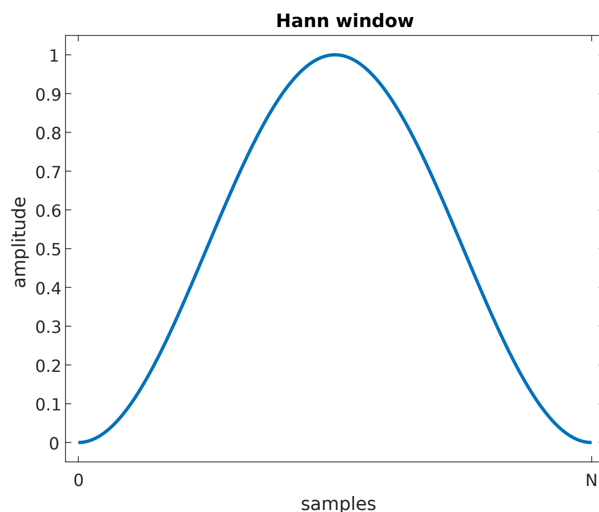
$$A(x) = a_0 + 2\sum_{n=1}^{\infty} a_n \cos\frac{n\pi x}{b}, \quad a_n \in R$$

kde koeficienty splňuji podmínku

$$1 = a_0 + 2\sum_{n=1}^{\infty} a_n$$

Na obrázku [1.2] lze vidět okénkovací funkci typu Hann. Obrázek [1.3] pak ukazuje vliv oné funkce. Levý horní graf reprezentuje nevhodně ořízlou sinusovku o 110 Hz,

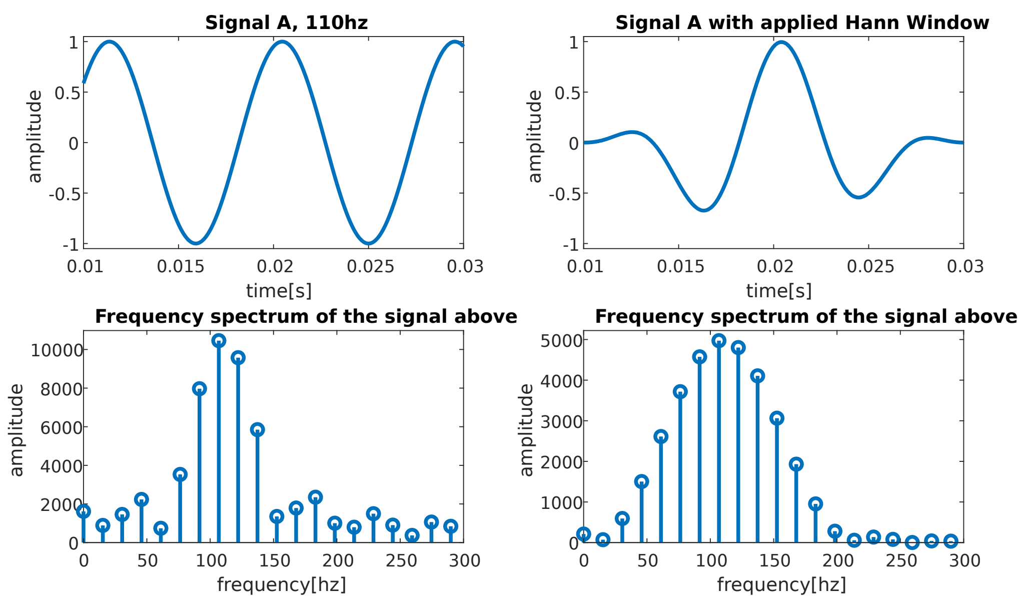
kdežto pravý onu sinusovku po aplikaci Hannova okna. Pod tím se nachází grafy ukazující frekvenční závislost odpovídajících signálů výše. Lze také vidět, že signál po aplikaci okna má a jemněji konvergující spektrum k základní frekvenci.



Obrázek 1.2: Hannovo okno

Poznámka 1.18. Diskrétní vektor w hodnot Hannova okna o délce N můžeme vypočítat dle následujícího předpisu [10]

$$w(n) = \frac{1 - \cos 2\pi nN}{2}, 0 \leq n \leq N$$



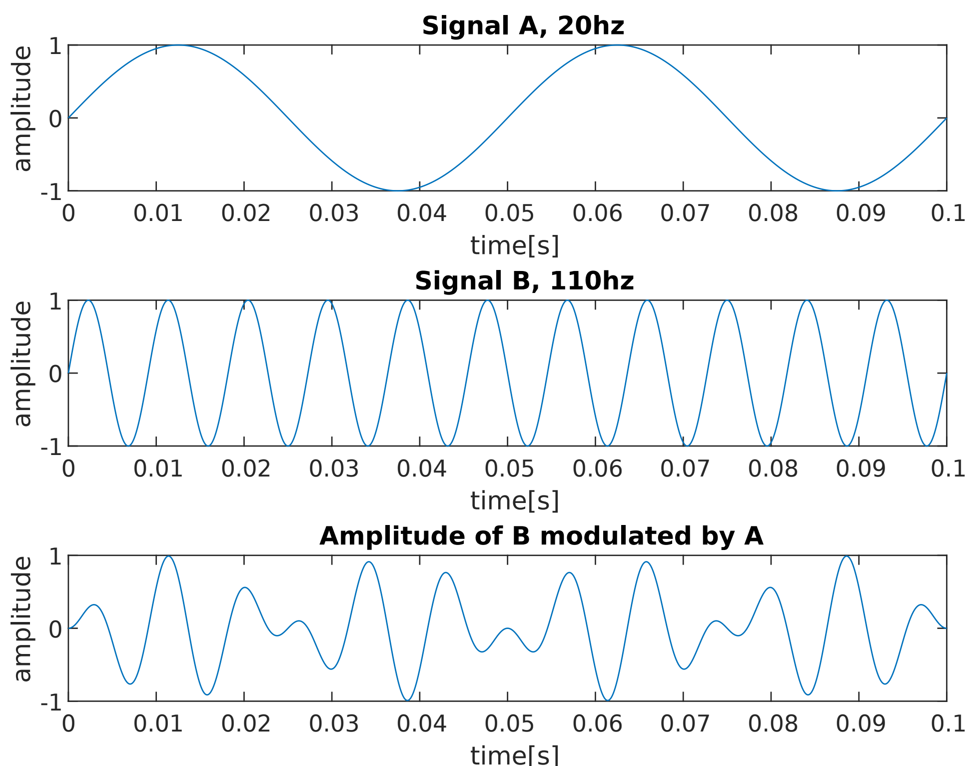
Obrázek 1.3: Vliv okénkovací funkce na frekvenci

1.6 Modulace

Protože se při implementaci některých efektu využívá modulace (viz phaser 4.4, vibrato 4.5), tak by bylo vhodné tento pojem zavést a osvětlit.[6, 11]

Definice 1.19. Modulací rozumíme proces, který v závislosti na nějakém periodickém signálu mění nějaké atributy nějakého systému.

Pod systémem se vetšinou rozumí signál, který se zpracovává. Jednoduchou ukázku modulace lze vidět na obrázku [1.4], kde sinusovka o 20 Hz pronásobením moduluje amplitudu sinusovky o 110 Hz. Modulovat se však nemusí na přímo pronásobením, ale například postupnou změnou nějaké proměnné v čase v závislosti na příslušné hodnotě periodického signálu. Například tak lze periodicky měnit parametry filtru a získávat v čase proměnlivou frekvenční odezvu.



Obrázek 1.4: Modulace amplitudy

2 Přehled Real-time audio aplikací

2.1 Digital audio workstation software

Digital audio workstation, zkráceně DAW, je software (nebo také obecně hardware či kombinace) sloužící pro náročné nahrávaní, zpracováni audia a tvorbu audio souboru.[12]

Dnešní populárni DAW softwary jsou relativně složitými aplikacemi nabízející velké množství funkcí pro práci jak s audio, tak i MIDI, podporu různých komunikačních protokolů mezi audio vstupem a výstupem (ASIO, WASAPI, atd.), či pluginů. I když nejsou přímo zaměřené na zpracovaní audio signálu v reálném čase, tak se jedná o velmi flexibilní nástroje, které se dají nastavit pro různé potřeby při zpracovaní audia. To má ale za důsledek složitá uživatelská rozhraní a vyžaduje dovednosti a schopnosti pro práci s konkretním software.

2.1.1 Audacity

Relativně jednoduchý bezplatný open-source software. Nabízí řadu funkcí pro práci s audiem, podporu různých standardu pluginů, jako VST, AU, LV2 a LADSPA a lze do něj psát i vlastni pluginy za pomoci jazyka Nyquist. Ke dnešnímu dni, ale nemůže oficiálně podporovat ASIO z licenčních důvodu. Má podporu real-time efektu, ale zatím s omezením na jeden efekt.[13, 14, 15, 16]

2.1.2 REAPER

Jedná se o profesionální software, který podporuje řadu standardů pluginů, VST, VST3, LV2, AU, DX, a JS. Jedná se o placený program, s distribuční politikou, kde je na 60 dní zdarma dostupná plnohodnotná verze programu, která se následně dá zakoupit za přijatelnou cenu. Jsou k dispozici dvě verze za 60$ a 225$, přičemž obě verze se liší pouze v podmínkách používaní, a ne omezení samotného software. Před koupí je uvedena verze do které bude platit bezplatná podpora. Program je velmi lehký, instalační soubor zabírá kolem 15 MB pro Windows a umožňuje spuštění z např. flash disku. Dá se také použit pro editaci videí. Dostupný jak na Winodws, tak i macOS a Linux.[17, 18]

2.1.3 Cubase

DAW, které je vyvinuté společnosti Steinberg (vyvinuli taky ASIO a VST). Momentálně nejnovější verzi je Cubase 12. Členi se do tři distribuci, Cubase Pro 12 Pro (579$), Cubase Artist 12 (329$), Cubase Elements 12 (99,99$). Verze se liší funkcionalitou v závislosti na ceně. Existuji také bezplatné verze, Cubase AI 12 a Cubase LE 12, které jdou v kompletů s některými zařízeními a nabízí velice omezenou funkcionalitu ve srovnání s placenými. 12tá verze přináší nový způsob aktivace programu, nyní není potřeba speciálního USB-eLicenseru a program lze normálně stáhnout a aktivovat. Program v základu nabízí velkou řadu před připravených nástrojů, knihoven vzorku atd. Je tedy připraveny na použiti hned z krabice. Podpora pluginu je však omezena jen na VST a AU. Umožňuje i editaci videa. Je dostupny pro Windows a macOS.[19, 20, 21]

2.1.4 Pro Tools

Program vyvinuty společnosti Avid. Program se nečlení do plnohodnotných verzi, které se dají na vždy jednotně koupit, jak je to například u Cubase. Misto toho je distribuovan na měsíční předplatné osnově. Existuji dva různé platební planý, Pro Tools (od 29.99$) a Pro Tools Ultimate (od 79.99$), které se liší ve funkcionalitě. Program lze plnohodnotně ozkoušet po dobu 30ti dnu. Jedná se o standard v nahrávaní a editaci audia se zaměřením na multi-track nahrávání. Je používán ve velkém počtu profesionálních nahrávacích studich. Avšak, neumí tak dobře pracovat s midi nastrojí jako třeba Cubase. Podporované jsou pouze RTAS a AAX pluginy. Dostupny jak pro Windows, tak i macOS.[22, 21, 23]

2.1.5 GarageBand

Jedna se o DAW aplikaci vyvíjenou společnosti Apple pro iOS zařízeni. Aplikace má velkou sadu nástrojů, které lze dotykově ovládat. Umožňuje také zapojení například kytary a použiti k hraní za pomoci před připravených zesilovačů. Je také možnost instalace pluginu třetích stran jako samostatných aplikaci z App Storu. Aplikace obsahuje i verzi pro macOS, která se v některých aspektech odlišuje od mobilní verze. Projekt lze pak třeba z mobilní verze přenést na macOS. Obě verze jsou dostupné zdarma.[24, 25]

2.2 Audio pluginy

Velkou roli při zpracování audia hrají různé pluginy. Jsou hlavní součásti při zpracovaní audia za pomoci moderních DAW, které nabízí podporu různých rozhraní.

2.2.1 Virtual Studio Technology

Zkráceně VST, jedná se o rozhraní navržené společnosti Steinberg. Je to asi nejpopulárnější rozhraní, které se používá pro implementaci pluginů, a je široce podporované.[26, 27]

2.2.2 Audio Units

Zkráceně AU, jedná se o rozhraní navržené společností Apple, které je částí operačního systému Mac X OS. Je tedy dostupné jen pro dané OS, ale také má větší potenciál, co se týče rychlosti.[28, 26]

2.2.3 Avid Audio eXtension

Zkráceně AAX, jedna se o rozhraní navržené společnosti Avid pro jejich DAW ProTools, ovšem jedná se tedy pouze o vnitřní formát ProTools.[26]

2.2.4 Guitar Rig

Jedná se o VST, AU a AAX plugin, který obsahuje stand-alone verzi a je dostupný pro Windows a macOS. Nabízí velkou řadu předpřipravených efektů a zesilovačů se zaměřením na zpracování kytarového vstupu. Umožňuje taky jak pouštět audio ve smyčce na pozadí, třeba k hraní backing-tracku, tak i dělat jednoduché nahrávky. Momentálně je nejnovější verzí Guitar Rig 6. Profesionální verze se dá koupit za 199€ a přechod ze starší verze stoji 99€. K ozkoušení programu je i demo verze nabízející plnohodnotnou funkcionalitu, která je dostupná po dobu 30 minut. Po vypršení času lze program opět vyzkoušet, ovšem uložení stavu provést nelze. Kromě profesionální verze je dostupná i Player verze, která je zdarma, ale má výrazně omezenu funkcionalitu.[29, 30, 31, 32]

2.3 Audio komunikační protokoly

K samotné komunikaci aplikace se zvukovou kartou většinou nedochází na přímo, ale za pomoci různých rozhraní. Pro real-time aplikaci je potřeba nízké latence, tedy obvyklá vysokoúrovňová rozhraní nejsou moc vhodná.

2.3.1 Windows Audio Session API

WASAPI, součásti Core Audio API ve Windows počínaje Windows Vista. jedná se o nízkoúrovňové rozhraní. Umožňuje komunikaci přes shared mode, kde dochází

k míchaní signálu se všemi ostatními aplikacemi ve Windows audio engine. Minimalní dosažitelná latence se tak pohybuje kolem 30 ms. Ve Windows 10 došlo k optimalizaci, a vnitřní buffery mohou být nastavené aplikaci, tedy lze dosáhnout až prakticky 10 ms latence. Kromě shared mode se může použít exclusive mode, kde se obchází Windows audio engine a dochází k přímé komunikaci s hardwarem. Zde lze dosáhnout nižší latence, avšak jiné aplikace nebudou mít přístup ke konečnému zařízení.[33]

2.3.2 Steinberg Audio Streaming Input Output

Zkráceně se označuje jako ASIO. Jedná se o komunikační protokol pro komunikaci aplikaci se zvukovými kartami vyvinutými firmou Steinberg. ASIO umožňuje obcházet standardní cestu audio komunikace Windows a komunikovat napřímo se zvukovou kartou, čímž umožňuje dosáhnout nejmenší možné latence.[34, 35, 36]

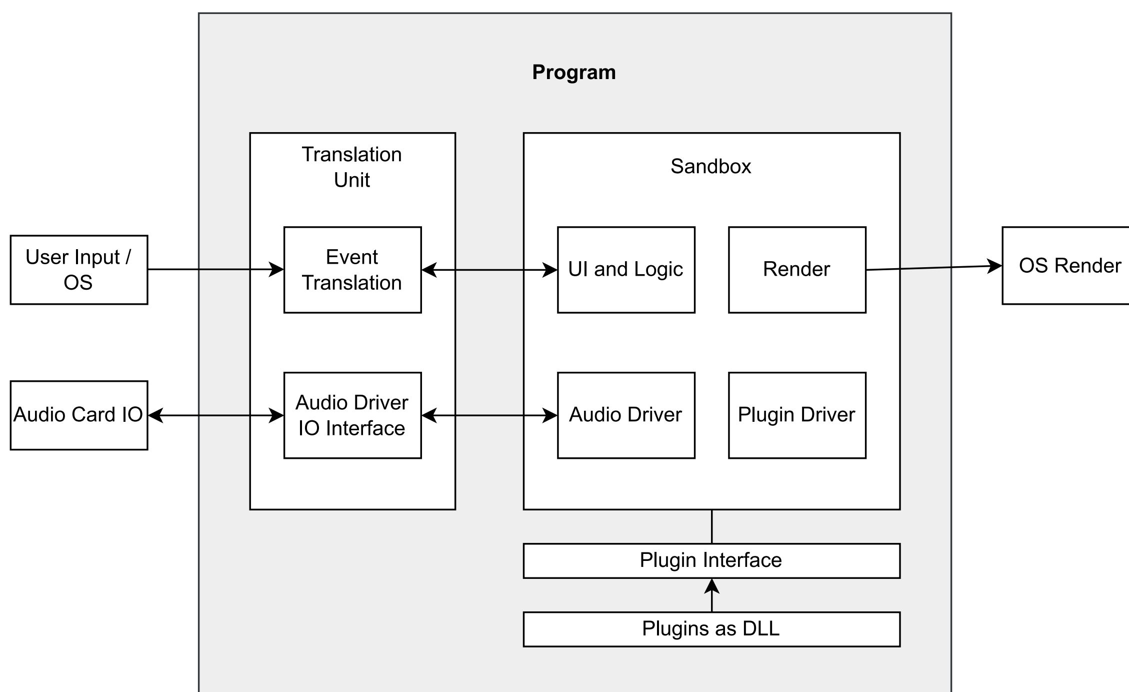
2.3.3 ASIO4ALL

Driver, který umožňuje používat ASIO rozhraní i pokud není podporováno zařízením. Slouží jako mezivrstva, která překládá požadavky přicházející z ASIO rozhraní aplikace na volání funkcí Windows pro komunikaci se zařízeními. Jelikož dnes i samotné Windows umožňuje komunikaci se zvukovými zařízeními napřímo za pomoci WASAPI, tak i u zařízení bez podpory ASIO lze někdy dosáhnout nižší latence než běžnými prostředky. [37, 38]

3 Praktický vývoj aplikace

Aplikace byla psaná v jazyce C++ přidržujíc se standartu C++14 za pomoci IDE Visual Studio 2019 pro operační systém Windows 10 za využiti Windows API. Ke komunikaci se zvukovou kartou se kromě Windows API použilo WASAPI a ASIO SDK. Při návrhu aplikace se dbalo na sníženi závislosti na operačním systému. Hlavní část aplikace je tedy v jakémsi sandboxu, kde veškerá komunikace probíhá za pomoci vnitřních stavů, struktur a rozhraní. Vstupy jsou pak překládané v mezi vrstvě na stavy vnitřní, které se dále posílají do sandboxu. Obdobně probíhá i zpětný překlad výstupů. Výjimkou je render, který na přímo využívá vnější funkce k vykreslení finálního bufferu pixelů. Schema návrhu je znázorněno na obrázku [3.1].

Takovýto návrh by měl umožňovat snadný přenos aplikace na jiné operační systémy, či dokonce prostředí bez operačního systému. V tomto případě by se musela přepsat pouze mezní vrstva. Klade to však vyšší nároky na výpočetní výkon.



Obrázek 3.1: Schéma návrhu programu

3.1 Uživatelské rozhráni

Základem uživatelského rozhraní je třída `Control` představující jeho libovolný prvek. Prvky rozhraní jsou navzájem propojeny do stromu, který vyjadřuje jejich hierarchii. Kořenem je instance `Window:Control`, která je pevně daná, existuje po celou dobu programu a je dostupna globálně jako statický atribut `Control`.

Vnější rozhraní `Control` je v zasadě představeno dvěma abstraktními funkcemi. Funkci `Control::draw` zajištující vykreslení komponenty a funkci `Control::processEvent` zajištující správu příchozích zpráv pro řízeni podnětů uživatele. Funkce jsou v základu implementované pro obecné použití. Ve většině případů by se měly používat implementované varianty, avšak, pokud komponenta potřebuje z důvodu výkonu, či funkcionality, změnit jejich chovaní, lze je přepsat.

V základu vykreslovaní probíhá tak, že se nejprve vykresli samotná komponenta a následně její děti. Při vykreslovaní dětí se nejprve zkoumá, zda chtějí byt vykresleny normálně v pořadí, jako poslední v rámci svého rodiče, nebo poslední v rámci celého stromu. Ve druhém případě se děti ukládají do prozatímního pole, které se vykreslí až na konci rodiče. Ve třetím případě se děti ukládají do globálního seznamu, který se vykresluje až po vykreslení základního stromu v kořeni ve `Window:Control::draw` (přepsána funkce `Control::draw`), kde následně dochází k jeho vyprázdnění.

Zprávy jsou posílané nejprve do kořenové komponenty, která je rozesílá dál stromem. Každá komponenta pak může libovolně zprávu zpracovat a poslat ji dle potřeby větví dál. Navíc, za pomoci výstupní hodnoty funkce `Control::processEvent`, dítě může žádat rodiče o ukončení propagace události do jeho jiných dětí. Tato zpráva se tedy může probublat až zpět do kořene a popřípadě ukončit šíření události po jejím prvním zpracování nějakou komponentou pokud je to třeba.

Všechny události jsou předem definované a zpracovaní každé z nich řeší konkretní callback funkce. Callback funkce komponenty se nastavuji přímo přes pointer. Je to nejjednodušší řešení, které má jednu zásadní nevýhodu — jedné události může byt přiřazen jen a pouze jeden callback. Sofistikovanější implementaci by bylo zabalení řízeni přidávaní callbacku události do nějaké třídy, která by je ukládala do seznamu a při ošetření události by je postupně volala. Ovšem, tato nevýhoda není tak výrazná, jelikož povětšinou jeden callback je dostačující a popřípadě ho lze využít jako jakéhosi prostředníka, který by se staral o správu vícero callbacků.

Jelikož původně tyto události pocházejí od operačního systému, tak před odesláním do vnitřního systému musí dojít k jejich překladu. Proto se při návrhu definice callback funkce použil praktický stejný popis jako ve winAPI. Usmyslilo se totiž, že pokud má byt aplikace co nejvíce oddělená od operačního systému a mít tedy nejprve mezivrstvu pro překlad události, tak je výhodné mít toto rozhraní shodné alespoň s jedním z existujících operačních systémů. Pak tedy, alespoň v jednom případě, se zjednoduší překlad. Proto definice parametru u většiny události jsou relativně podobné definicím ve winAPI.

3.2 Obsluha audia

Obsluha audia v programu byla řešena na způsob singletonu. Veškerou komunikaci zajišťoval jeden audio ovladač realizovaný jako namespace `AudioDriver`. `AudioDriver` spravoval několik různých instancí třídy `IAudioIO`. `IAudioIO` definuje rozhraní audio vstupu a výstupu. Implementace `IAudioIO` mohly například komunikovat s různými API reálných audio driverů, či třeba pouze soubory, nebo sériovými porty. Navenek `AudioDriver` tedy nabízí relativně podobné rozhraní, které umožňuje změnu konkrétní instance `IAudioIO`, a také se jedná o místo, kde se spravuje zřetězené zpracovávaní pluginů a jejich připadná obsluha.

Samotný zpracovávající řetězec a správa pluginů byla řešena následovně. Základní program načte při startu pluginy z DLL souborů a uloží si je globálně jako tzv. vzorkové pluginy. Pokud dojde k otevření nějakého pluginu, tak se přidá odpovídající vzorkový plugin do `AudioDriver`, kde bude vytvořena jeho kopie, která se následně přidá do aktivních pluginů. Tímto je umožněno otevření několika stejných pluginu naráz. Pokud by se přidávali pouze ukazatele na vzorkové pluginy, tak by pluginy měli sdílenou paměť, a tedy by nebyla možnost individuální konfiguraci. Aktivní pluginy se pak při zpracování sekvenčně procházely a každý dostával na vstup výstup předchůdce.

Implementované byli **ASIO**, **WASPI**, **LEGACY** a **FILE** `IAudioIO`.

3.2.1 ASIO rozhrání

Implementace ASIO rozhraní vycházela ze základní ukázky přiložené v ASIO SDK [36], která se dá stáhnout na oficiálních stránkách Steinbergu. Základem rozhraní jsou callbacky. Vyznačují se dvě takové funkce. Jedna pro zpracování samotných bufferu a druhá pro reakci na změny stavů (např. změna vzorkovací frekvence). Druhá funkce nebyla implementovaná, jelikož po startu programu již nešlo změnit stav driveru. Jakákoliv změna nastavení se do programu nepromítla a nedocházelo k volání funkce, tedy nebylo možné ověřit připadnou implementaci. Příčina této skutečnosti vyšetřená nebyla.

Při samotném zpracování dat se má v ideálním případe poskytnout implementace pro všechny datové typy, které může zařízeni dle popisu rozhraní použit ke komunikaci. Avšak k dispozici byla jen jedna zvuková karta, tedy implementace ošetřila pouze typ, který byl pro ni relevantní (int32 jako little-endian). Samotná data jsou seskupená jako pole struktur pro každý kanál. V poli nejprve jdou vstupní a následně výstupní kanály. Každá struktura pak obsahuje pole samotných bufferů a informace k nim. Jako vstupní parametr do funkce se pak předává index aktivního bufferu.

3.2.2 WASAPI rozhrání

Jako alternativa k ASIO byla implementovaná podpora WASAPI. K reprezentaci audio formátu WASAPI používá struktury `WAVEFORMATEX` a `WAVEFORMATEXTENSIBLE`. Jedna se o reprezentaci hlavičky WAVE souboru. Tedy formát je relativně univerzální a v teorii může podporovat velké množství datových formátů, i komprimovaných. Což ale má za nevýhodu ne moc jednoznačnou implementaci. Jako i v případe ASIO se implementovala pouze varianta, která byla využívaná zařízením, a to float32. Data v bufferech jsou uložena obdobně jako ve WAVE souborech. Jedná se o souvislé pole vzorku, které je členěno do stejně velkých skupin. Každá skupina pak obsahuje vzorky všech kanálu pro daný index. Implementace byla provedena za pomoci eventu, avšak lze si hlídat časovaní ručně a odebírat a přidávat data kdykoli. Bohužel, provést řádnou implementaci se nepodařilo. Po inicializaci se občas může projevit de-synchronizace vstupu a výstupu. Dokumentace Microsoft sice obsahuje ukázky základních implementaci, avšak ony se vždy tykají buď pouze vstupu, nebo pouze výstupu, ale nikde nedisponuje ukázkami souběžné implementace a řešení problému synchronizace. Tedy, prozatím, i když WASAPI funguje, není moc stabilní. Ve snaze zaručit větší stabilitu, opustila se podpora exclusive modu. Vždy se tedy používá výchozí datový formát v shared modu. Sice to neřeší problém, ale alespoň snižuje ostatní podněty, které by mohly byt vyvolané dynamickou režii různých formátů.

3.2.3 Legacy rozhrání

Jelikož implementace WASAPI nebyla úplně úspěšná, a ani samotné API nemusí byt podporované na některých systémech, tak bylo rozhodnuto implementovat nějaké tzv. legacy rozhraní, které by mohlo byt jakou si záložní cestou na úkor latence. Rozhraní bylo implementované za pomoci `waveIn` a `waveOut` API. K popisu formátu dat se využívají stejné struktury jako u WASAPI. Zde již byly implementované všechny základní datové formáty — int8, int16, int32, int64, float32, float64. Opět, jako i u WASAPI se zde volil vždy výchozí formát dat nabízeny operačním systémem.

3.2.4 Souborový vystup a výstup

K tomu, aby mohla byt ověřena funkčnost pluginu jinak než empiricky, se přes rozhraní `IAudioIO` implementoval souborový vstup a výstup. Tedy program načte vstupní soubor, dávkově ho zpracuje, simulujíc tím real-time zpracovaní, a výsledek zapíše do souboru výstupního. Cesty jak k vstupnímu, tak i výstupnímu souboru jsou předem definované v konfiguračním souboru. Je tedy možno za běhu aplikace libovolně měnit vstupní soubor zanechávajíc stejný název. Byla implementovaná podpora dvou formátů dat. Prvně, surový formát, tedy pouze binární data jako posloupnost floatů. Druhotně, WAVE formát, kde se pro vstupní signály implementovaly 8 (unsigned), 16, 24 (bez paddingu), 32 bit celočíselné a 32 a 64 bit s plovoucí řádkou formáty. Výstupní formát se zvolil jednotný, a to celočíselný 32 bitový.

3.3 Předvolby

Důležitým bylo poskytnout uživateli možnost ukládat si nastavení pluginů a jejich pořadí ve formě předvoleb, neboli tzv. presetů. Bylo rozhodnuto ukládat data v čitelné formě, jako prostý text v ASCII formátu, aby bylo možno jednoduše měnit, číst, či vytvářet presety nezávislé na programu.

Hlavním dilematem byla identifikace pluginu. Plugin jako takový může být napsán kýmkoli a po pouhém položení do příslušné složky načten programem. Tedy nějaké vnitřní id by nebylo použitelné, jelikož by jednoduše docházelo ke kolizi. Hlavně proto, že by tato informace nemohla byt rozhraním vyžádaná a většinou by byla nastavena na nějakou standardní hodnotu, např. nulovou, nebo by nebyla inicializovaná a byla by při každém startu programu nejspíš jiná.

Možnou variantou by mohlo byt použiti hashe binárního souboru pluginu, avšak i zde je stále možnost kolize. Kolize samotná však není takovým problémem jako skutečnost, že plugin bude představen jako nečitelná sleď znaků. Nejedná se tedy o intuitivní reprezentaci pluginu ze stránky uživatele, čímž se jaksi ztrácí význam uchování dat v čitelné formě.

Více intuitivní a jednodušší variantou by bylo použiti jména souboru, jelikož to zaručuje jednoznačnost názvu. Ovšem, to samé jméno souboru nemusí zaručit jednoznačnost souboru samotného. Aby tato skutečnost byla ošetřena, rozhodlo se pro použiti dvojitého id — jméno pluginu jako primární a jméno souboru pluginu jako záložní. Nejprve se plugin identifikuje dle názvu a pokud dojde ke kolizi, tak se použije jméno souboru. Jelikož záložní id je jednoznačné, tak se takovým stává i celý předpis. Je tedy jednoduché určit, jak se program zachová v případe konkretního presetu. Samozřejmě tato možnost není neprůstřelná a pod stejný preset můžou sedět různé pluginy, avšak pro běžné používání to vypadá jako dobrý kompromis.

Formát samotného preset souboru vypadá následovně

```
preset_name
<plugin_1_specification>
<plugin_2_specification>
.
.
.
<plugin_n_specification>
```

Preset tedy začíná jeho názvem a dále, vždy na novém řádku, sledují specifikace pluginů, přičemž jejich pořadí odpovídá pořadí při načítaní do programu (ze shora dolů). Jak název, tak i jakákoliv specifikace, může být, kromě nutného oddělovacího znaku nového řádku '\n' (ASCII hodnota 10), oddělen libovolným počtem bílých znaků.

Samotna specifikace pluginu vypadá následovně,

```
name:filename:state:controls_count[value_1, value_2 ... value_n]
```

kde jednotlivé parametry jsou oddělené dvojtečkou ':' (ASCII hodnota 58). Specifikace počíná `name`, které vyjadřuje jméno pluginu sloužící jako primární id, a následné `filename`označuje jméno souboru pluginu, sloužící jako sekundární id (jménem souboru se myslí pouze název, bez cesty a rozšíření). Dále sleduje `state`, které vyjadřuje stav pluginu jako celé číslo (0, vypnutý; 1, zapnutý; 2, pouze levý kanál; 3, pouze pravý kanál). Za ním je `control_conut` určující celým číslem počet ovladačích prvku pluginu. Následně v hranatých závorkách (znaky '[' a ']', ASCII hodnty 91 a 93) je posloupnost hodnot jednotlivých ovládacích prvků oddělených čárkou. Hodnoty jsou vyjádřené jako čísla s plovoucí čárkou, kde desetinná tečka se značí symbolem '.' (ASCII hodnota 49). Všechny parametry můžou byt obklopené libovolným počtem bílých znaků, kromě znaku konce řádku, který slouží pro oddělení specifikací samotných.

3.4 Návrh rozhrání pluginu

Ke komunikaci programu s pluginy bylo nutné navrhnout rozhraní. Primárním cílem byla jednoduchost. Avšak, rozhraní by mělo byt schopno zajistit alespoň následující klíčové vlastnosti

- možnost konfigurace
- možnost využívání vice stejných pluginů současně
- možnost uložení data mezi zpracovávajícími se dávkami
- zbavit uživatele nutnosti řešit vykreslení uživatelského rozhraní
- pojistit alespoň dvoukanálovou podporu

Jak již bylo naznačeno 3.2, aby byla možnost použití několika stejných pluginů současně, v klientském programu se pluginy musí kopírovat. Samozřejmě, pokud se vyžaduje možnost asociace dat s pluginem, tj. vlastně možnost konfigurace.

Pokud jde pouze o konfiguraci, tak v základu je dostačující pouze hodnota ovládacího prvku, která by se četla vždy při zpracování. Avšak, chtěná by byla možnost předvýpočtu a ukládaní stavu pro následující zpracování. Jelikož formát dat je od pluginu k pluginu velmi odlišný, nemá cenu se ho snažit předdefinovat. Uživateli tedy bude umožněno si dle svých potřeb alokovat paměť a ukazatel na ni mu bude poskytnut jako součást vstupů funkcí rozhraní. I když takovýto přístup nemusí vypadat moc příjemně pro uživatele, tak onu paměť lze v přehledné formě reprezentovat v jazyce C strukturou. Tedy práce s ní se stává relativně jednoduchou bez řešení offsetu a s fungující nápovědou v různých IDE.

Samozřejmě, některá data mohou byt závislá na určitých nekonstantních parametrech samotného programu (např. délka bufferu je v sekundách, tedy závislá na vzorkovací frekvenci). V případě jejich změny je nutno umožnit uživateli patřičným způsobem reagovat. Jak alokace, tak i případná realokace je řešena přes jedinou funkci `IPlugin.init`, která kromě ukazatele na ukazatel paměti uživatele dostává i relevantní informace o stavu programu. Tato funkce je volaná jak při první inicializaci pluginu, tak i při případných změnách stavu programu.

Ke konfiguraci, kromě paměti, je také potřebné uživatelské rozhraní. Základem uživatelského rozhraní je struktura `PluginUIHandler`, která je převážně používaná samotným programem. Ta obsahuje ukazatel na pole struktur `PluginControl`, které reprezentují jednotlivé prvky uživatelského rozhraní. Součástí rozhraní jsou i funkce, které tyto struktury inicializuji na standardní hodnoty. Uživatel je pak fakticky vyžádán jen o určení typu ovládacího prvku. Ostatní hodnoty již mohou být nastaveny dle potřeby. Uživatel si může kromě funkčních hodnot nastavit i vykreslovací hodnoty jako barva, typ pozadí atd., to jsou ovšem ze strany rozhraní jen doporučené hodnoty a program k nim nemusí nahlížet.

```
PluginUIHandler* const uihnd = buildPluginUIHandler();
uihnd->controls[0]->backgroundColor = 0xFFEA6363;
uihnd->controls[0]->fillType = PFP_DOTS;

PluginControl* const gainKnob = addControl(uihnd, PCT_KNOB);
gainKnob->MAX_VALUE = 30.0;
gainKnob->value = 0.0;
gainKnob->color = 0xFF000000;
gainKnob->label = "Gain";
gainKnob->eChange = &gainChange;
```

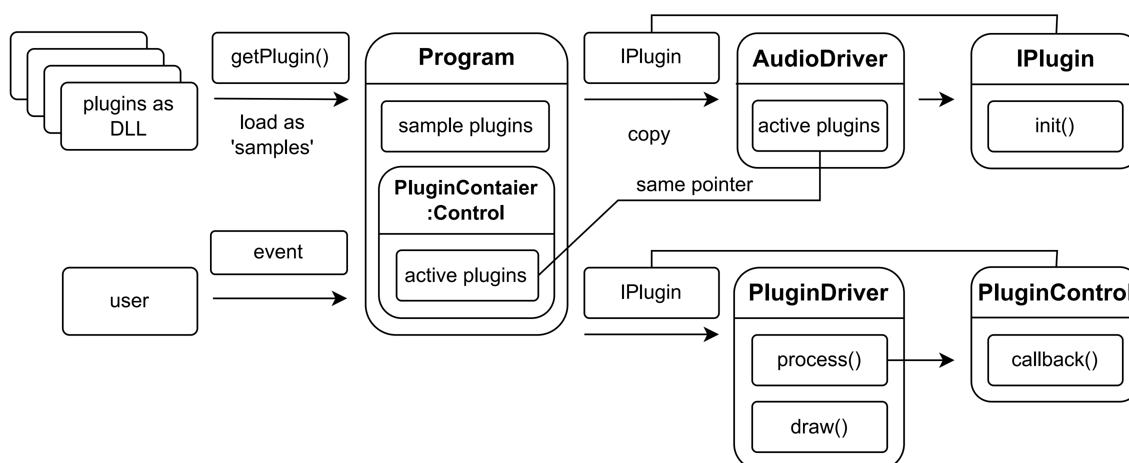
Zdrojový kód 3.1: Vytvoření uživatelského rozhraní pluginu s jednou kontrolkou

Jelikož některé konfigurační hodnoty nemusí napřímo odrážet reálné hodnoty při zpracování signálu (například hodnota zesílení se vyjadřuje v decibelech, ale při výpočtu se musí převést na amplitudu), nebo při jejich změně bude potřeba provést i jiné akce, není moc vhodné jejich hodnoty počítat pří zpracování každého bufferu. Rozhodlo se tedy řešit tyto změny callbacky, které by byly volané v hlavním (vykreslovacím) vlákně, tedy paralelně vůči vláknu zpracovávajícímu audio. To může být v některých případech nevýhodné, ovšem, pokud by byla synchronnost nutná, tak vždy lze callbacky nepoužívat a provádět výpočty až při zpracování.

Samotná funkce pro zpracování signálu byla nakonec navržena tím nejjednodušším způsobem. Funkce dostává vstupní buffer, výstupní buffer, délku bufferu a ukazatel na uživatelem alokovanou paměť. V takovémto případě však není umožněno zpracování několika kanálu. Je to proto, že předávání bufferů všech kanálu by ztížilo implementaci pluginu. Uživatel by pak například musel řešit — co když se počet bufferů změní, může li se spolehnout na to, že první buffer bude vždy levý a druhy pravý, atd. Tato dilemata nejsou moc chtěná člověkem, který by si chtěl zkusit napsat jednoduchý plugin a pohrát si se zvukem, přičemž mít fungující kus kódu. Tedy, bylo rozhodnuto zbavit uživatele těchto dilemat a řešit to na úrovní programu, tedy

využít toho, že lze použit stejný plugin několikrát. Dle potřeby by se pak v programu mohl volit kanál pro zpracování konkretního pluginu, což bylo shledáno za důstojný kompromis.

Jelikož program je stavěn pro zpracování v reálném čase, tak jednotlivé dávkovací buffery pokrývají řady milisekund, kdežto uživatelské buffery se mohou měřit v sekundách. Tedy by bylo z praktického důvodu vhodné zajistit, aby uživatel mohl mít možnost si alokovat dostatečně velký buffer, u kterého by mohl počítat, že bude větší než dávkovací buffery. Tím by se jak zjednodušila, tak i zrychlila, rutina zpracování. Program tedy zaručuje maximální délku svých bufferů, která je zasílaná jako jedna z položek do inicializační funkce.



Obrázek 3.2: Plugin a program

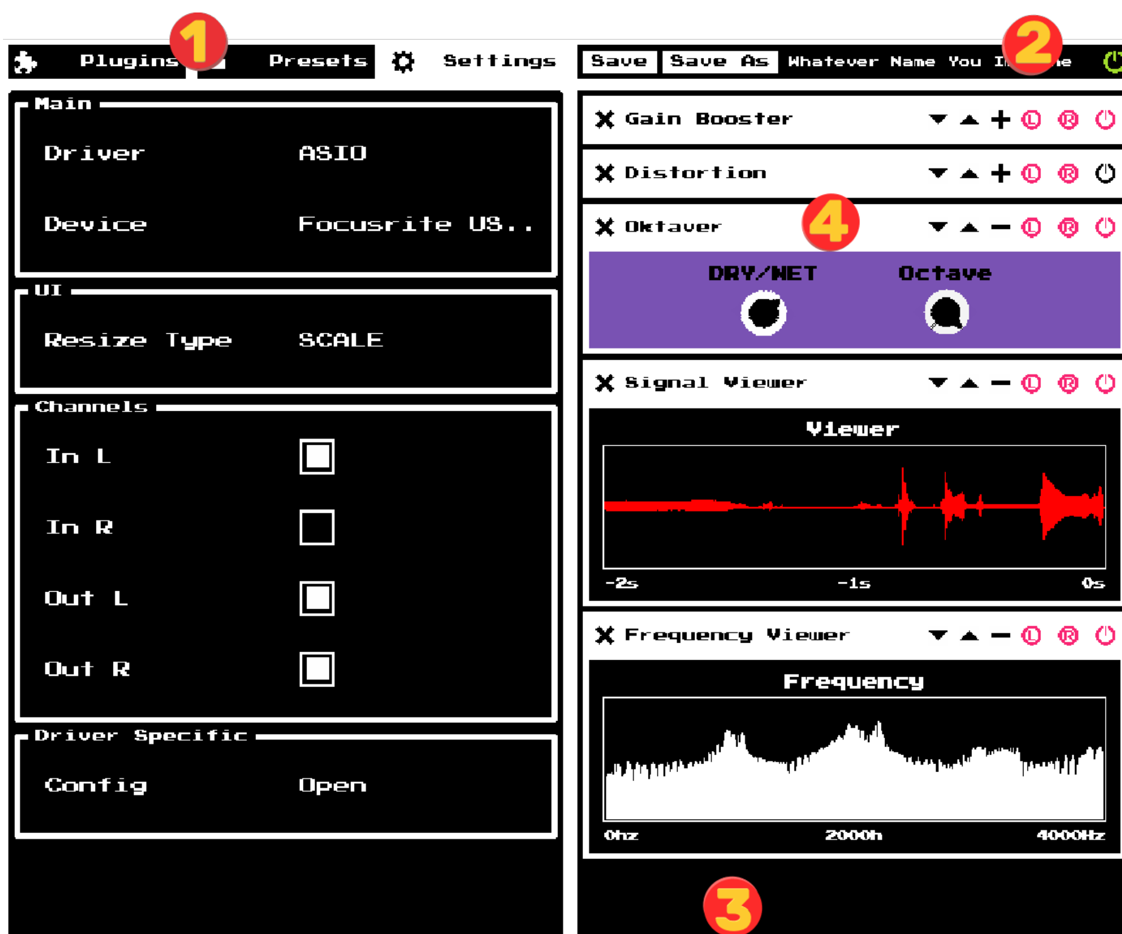
Rozhraní se obohatilo i o možnost jednoduchého vykreslování signálu. Rozhraní tedy definuje strukturu `PlotInfo` představující jednoduchý protokol pro komunikaci s programem. Struktura obsahuje pole pro definici vzorkovací frekvence, ukazatel na buffer, který je vnímán jako kruhový, jeho délku a index posledního vzorku. Navíc také obsahuje i ukazatel na void, který slouží pro připadně cachování dat ze strany programu. Realizace byla provedena za pomoci uživatelských prvků `PluginControl`. Vytvořili se dva speciální typy `PCT_SIGNAL_VIEWER` a `PCT_FREQUENCY_VIEWER`, pro vykreslení amplitud, respektive frekvencí. Program pak, při jejich vykreslení, interpretuje počátek uživatelské paměti jako strukturu `PlotInfo`. Uživatel pak může v zpracovávající funkci libovolně transformovat data a ukládat je do kruhového bufferu k vykreslení. Ten pak bude průběžné vykreslován kontrolkou. Uživatel tak třeba může napsat plugin pro zobrazení energie signálu. Navíc také, uživatel může dělat i jiné výpočty a provádět úpravy signalu. Jelikož samotné okno se chova jako prvek rozhraní, a ne typ pluginu, tak je umožněno i přidávat jiné konfigurační prvky. Jak amplitudové, tak i frekvenční vykreslování lze vidět na obrázku [3.3].

Plugin, jak již bylo naznačeno, jako takový nenabízí žádné funkce pro jeho vykreslení, jen samotné hodnoty. Veškeré funkce pro řízení události, logiky ovládacích prvků a vykreslení, jsou obsažené v PluginDriveru (viz. obrázek [3.1]), který je součástí programu.

I když samotné rozhraní bylo navrženo a implementováno v jazyce C, pluginy jsou kompilované do DLL souborů s jednou jedninou funkci vracející ukazatel na samotný plugin představeny strukturou `IPlugin`. Tedy, teoreticky, by k jejich implementaci mohl byt použit i jiný jazyk, pokud by kód šel zkompilovat do odpovídající binární podoby. Navíc, samotné DLL by šlo zaměnit za nějakou jinou obdobu dynamického linkovaní (například shared object u linuxu), a samotný kód pluginy by zůstal stejný, jen by se jinak namapovala jedna jediná funkce. Tedy, DLL třeba lze i úplně vynechat a pluginy popřípadě zkompilovat vestavěně do programu.

3.5 Výsledný stav

Na obrázku [3.1] lze vidět výsledný stav uživatelského rozhraní programu. Program je primárně rozdělen do dvou vertikálních bloku. V levo nahoře (1) je menu umožnující výběr obsahu bloků pod ním. Aktuálně je otevřeno nastavení. Další dvě možnosti jsou nabídky s výběrem pluginu, nebo presetu, obě realizované jako jednoduchý abecední seznam. V právo nahoře (2) se nachází panel s tlačítky save a save as pro ukládaní momentálního stavu tzv. batohu pluginů (3). Panel taky uvádí název momentálně otevřeného presetu a taky disponuje tlačítkem pro zapnutí či vypnutí zpracovaní audio signálu. Horní panel pluginu (4) umožňuje posouvat pluginy ve frontě nahoru (zpracovaní má pořadí od shora dolu), dolu, skrýt a zobrazit tělo, vypnout nějaký kanál pro zpracování, či samotný plugin. Tlačítko pro odstranění pluginu se nachází zleva, odděleně od funkčních tlačítek, aby nemohlo dojit k nechtěnému překliku.



Obrázek 3.3: Uživatelské rozhraní

4 Implementované audio efekty

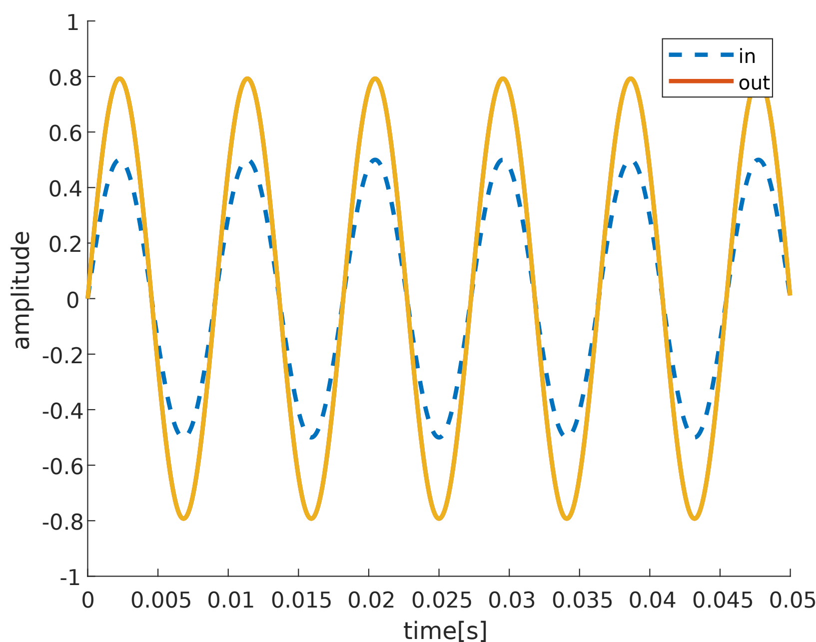
Přehled, popis a testování implementovaných audio efektů přes navržené rozhraní. K zpracování a analýze výstupu se použilo prostředí MATLAB.

4.1 Zesílení

Jedná se o nejzákladnější efekt, kdy signál je násoben kladnou hodnotou, čímž dochází ke zvýšení amplitudy, a tedy i akustickému zesílení signálu.

Zesílení signálu se zpravidla vyjadřuje v decibelech, tedy je vhodné je používat i pro konfiguraci a v při výpočtu převádět na amplitudu.

Srovnání vstupu (sinosovky o frekvenci 110 Hz) a výstup pluginu při aplikaci zesílení o 4 dB lze vidět na obrázku [4.1].



Obrázek 4.1: Zesílení o 4 dB

4.2 Zkreslení

Obecně se jedná o libovolnou změnu tvaru signálu vůči originálu. V hudbě se tímto pojmem povětšinou označuje efekt založený na oříznutí signálu. Například při velkém zesílení, kdy amplituda signálu překročí možný rozsah, čímž dojde ke ztrátě informace a změně signálu. I když tento efekt je ve většině případu nechtěný, tak ho lze využit k tvorbě agresivního, bzučícího zvuk, který se dá využít například při zpracování kytarového signálu.[39, 40]

V reálných implementacích ovšem nedochází jen k umělému ořezávaní signálu, ale i k jeho filtraci různými filtry v různých fázích zpracování, což ve výsledku vytváří charakterní zvuk nějakého produkčního modelu.

Oříznout signál lze různými způsoby, které můžou být vyjádřené za pomoci funkce. Nejzákladnější takovou funkci je Hard clip, která na základě pevně daných mezí ořízne signál.

Definice 4.1. Nechť $x \in R$, pak Hard clip definujeme následovně [41]

$$f(x) = \begin{cases} -1, & x \leq -1 \\ x, & -1 < x < 1 \\ 1, & x \geq 1 \end{cases}$$

Další základní funkci je Soft clip, která se vyhlazuje přechody před oříznutím.

Definice 4.2. Nechť $x \in R$, pak Soft Clip definujeme následovně [42]

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

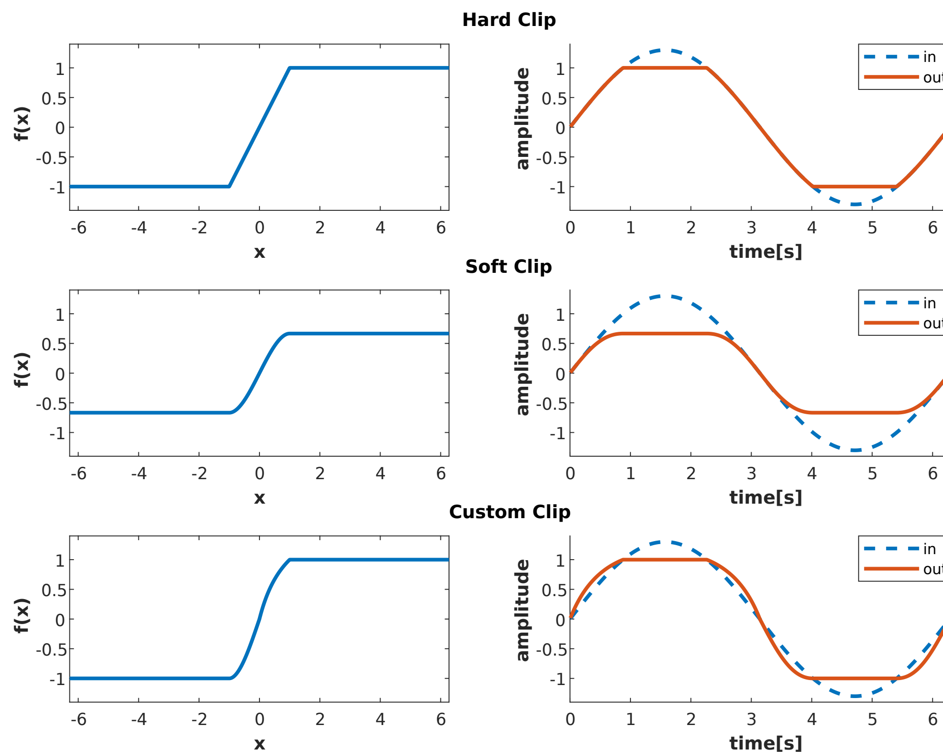
Funkce však může byt volena relativně libovolně dle potřeb. Převážné by se měla uvažovat monotonní spojitá funkce, aby byl zajištěn plynulý přechod. K tvorbě takové funkce lze využit např. funkce arctan, sin, $\frac{1}{x}$, exp atd.

Při implementaci se použila následující funkce.

$$f(x) = \begin{cases} -1, & x \leq -1 \\ -1 < x < 0, & \frac{3}{2}(x - \frac{x^3}{3}) \\ 0 \geq x < 1, & \frac{-2}{x-1} + 2 \\ 1, & x \geq 1 \end{cases} \tag{4.1}$$

Zde lze vidět, že se již nejedná o funkci lichou, takovéto chování se označuje za asymetrické [6].

Srovnání funkcí lze vidět na obrázku 4.2. Levý sloupec ukazuje tvar samotných funkcí pravý jejich příslušnou aplikaci na příkladě sinusovky.



Obrázek 4.2: Ořezávací funkce

Při implementaci byl využíván následující zdroj [43]. Signál byl nejprve zesílen 36 až 70 krát, dle konfigurace. Následně došlo k filtraci high-pass filtrem o 33 Hz a jeho následnému oříznutí. Tím se docílilo silnějšího oříznutí vyšších frekvencí, které byly po filtraci více zesílené. Potom se signál rozdělil na dvě cesty. Jedna procházela low-pass filtrem o 234 Hz. Druhá high-pass filtrem o 1063 Hz. Cesty se následně míchaly dle dynamicky nastavitelného poměru. Tím bylo docíleno možné změny tónu. Před výstupem byl signál popřípadě zesílen. K filtraci se použily biquad filtry 4.9, kde parametr Q byl volen jako 0.05.

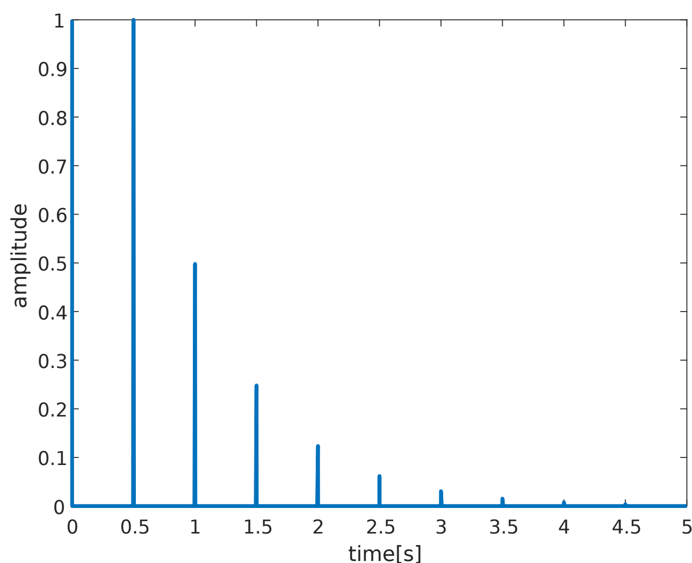
4.3 Delay

Efekt spočívající v opoždění signálu v čase. Opožděný signál se může míchat se signálem původním, čímž dochází k tvorbě ozvěny (echa). Vstupní signál se může také míchat se signálem výstupním, tzv feedback, čímž se dá docílit vícenásobné ozvěny, která by, pokud by výstupní signál přiváděný na vstup byl zeslabován, postupně uhasínala.[6]

K implementaci se dá použít kruhový buffer, do kterého by se zapisovaly přicházející vzorky. Při zpracování vzorku by se pak z bufferu vybral příslušný vzorek v závislosti

na potřebném zpozdění. Feedback by se dal přidat jednoduše zvětšením délky kruhového bufferu na dvojnásobek. Pak při zpracování vzorku lze získat jak prvek s jednotným, tak i dvojnásobným zpožděním. Alternativně se může jít přímo a použít druhý kruhový buffer na ukládaní výstupu. Při implementaci se použila první varianta.

Výstup pluginu pro jednotkový impulz s délkou odezvy 0.5 s a na polovinu zeslabeným feedbackem lze vidět na obrázku [4.3]. Je vidět, že prvotně se signál přehrál beze změny, a následně, po půl sekundě, se přehrál znovu. Potom již docházelo k přehrávaní zeslabených kopií, které vznikaly z feedbacku a každá následující měla dva krát nižší amplitudu.



Obrázek 4.3: Delay, odezva jednotkového impulzu

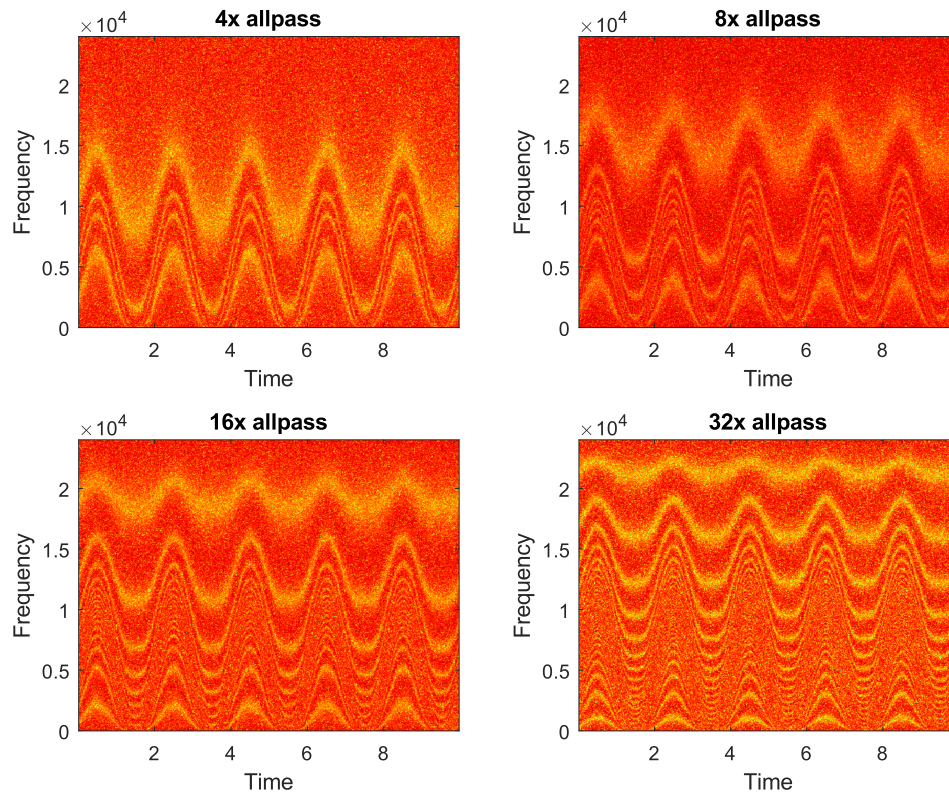
4.4 Phaser

Jedná se o efekt, kdy dochází k míchaní originálního signálu se signálem posunutým ve fázi, přičemž změna fáze je modulována nízkou frekvencí (většinou do 20 Hz). Dochází tedy k proměňujícím se v čase zeslabení, či úplnému rušení některých frekvenci. Výsledný zvuk pak má jakýsi pulzivní protékající charakter.[6]

Implementován byl několika za sebou jdoucími all-pass filtry, které měnili fázi signálu. Výstup posledního all-pass filtru se pak přičítal k originálnímu vstupnímu signálu. Frekvence all-pass filtrů byla modulovaná funkcí sinus v rozmezí 100 Hz až 10 000 Hz.

Jako testovací signál se použil bílý šum, jelikož jsou v něm obsazené relativně všechny frekvence, což umožňuje názorné zobrazení změn ve frekvenčním spektru.

Výstup pluginu při počtu 4, 8, 16 a 32 za sebou jdoucích all-pass filtrů s poměrem jedna ku jedné originálu a fázově modulovaného signálu lze vidět na obrázku [4.4]. Je vidět, že se ve výstupním signálu periodicky mění frekvence. Navíc, se zvýšením počtu all-pass filtrů roste i počet zářezů ve frekvenčním spektru, což vypadá jako správné chování[44].



Obrázek 4.4: Phaser, odezva bílého šumu

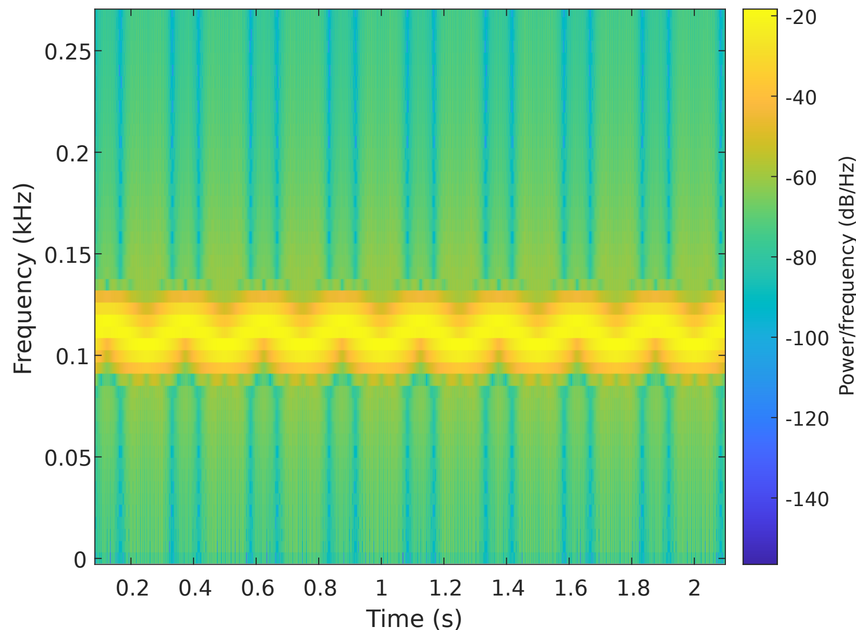
4.5 Vibrato

Vibrato je efekt, který periodicky zvyšuje a zpětně snižuje tón signálu. Je charakterizováno velikostí periody a rozmezím deviace tónu.[45]

Implementace byla provedena za pomoci modulace indexu krátkého delay bufferu. Odpovídající vzorek bufferu pak byl přímo poslán na výstup. Dochází tedy k jakému si neustálému zpomalení a zrychlení přehrávání, a tím tedy ke změně tónu. K modulaci se použila funkce sinus. Změnou kroku fáze sinu se mohlo docílit změny periody kmitu vibráta a změnou délky bufferu se mohl změnit rozsah deviace tónu.

Nevýhodou takové implementace je slyšitelné roztržení zvuku při použití většího zpoždění než 0.1 s. Navíc, se zvýšením zpoždění roste i přítomnost vyšších harmonických.

Na obrázku [4.5] lze vidět vystup pluginu pro sinusovku o 110 Hz. Délka zpoždění byla nastavena na 1 ms a frekvence kmitu na 4 Hz. Je vidět jak přítomnost vyšších harmonických, tak i postupné periodické změny frekvence signálu 4 krát za sekundu.



Obrázek 4.5: Vibráto, spektrogram výstupu sinusovky 100 Hz

4.6 Chorus

Efekt vznikající při sloučení několika signálů s přibližně stejnou výškou tónu v přibližně stejný čas. Efekt tedy slučuje kopie vstupního signálu lehce se lišicích ve výšce s různým zpozděním pohybujícím se v řádu milisekund. Takto lze napodobit současnou hru několika nástrojů.[6]

Implementace byla založena na vibrato efektu 4.5. Nejprve se definovala struktura reprezentující vibrato. Následně se definovalo pole s šesti instancemi oné struktury, kde každá instance byla inicializovaná lehce odlišnými hodnotami. Při zpracování se data průběžné ukládala do jednoho kruhového bufferu. Načež se postupně procházely všechny instance a dle jejich parametru se vybíral patřičný prvek bufferu. Vybrané prvky se průběžně sčítali a výsledek byl nakonec přičten k originálnímu vstupnímu vzorku.

4.7 Octaver

Efekt, kdy dochází k posunuti tónu signálu o oktávu, případně několik oktáv, nahoru či dolů. Většinou slouží k obohaceni zvuku a tudíž se prakticky vždy protíná s originálním signálem.[46, 6]

Implementace byla provedena za pomoci phase vocoderu, viz poznámka 4.3. Vstupní vzorky se zpracovávaly s určitým krokem N, tedy vždy, když se do kruhového bufferu načetlo nových N vzorků, tak začalo samotné zpracování. Zpracování se provádělo na okně o M minulých vzorcích, kde $M > N$. Zpracovávané úseky se tedy navzájem překrývají. Navíc, na každý úsek se vždy aplikovala Hannova okénkovací funkce 1.18. Jako okno, tak i překrývaní sloužilo ke sníženi vlivu roztržení signálu na hranicích okna.

Při samotném zpracování okna nejprve docházelo k jeho rozkladu na fáze a magnitudy, úpravě fáze, zpětné resyntéze a nakonec resamplingu. Tedy, nad oknem bylo provedeno reálné K bodové FFT, kde $K >= M$, přičemž okno se doplnilo odpovídajícím počtem nul. Z výsledku se získaly magnitudy a fáze, viz 1.6. Fáze se následně upravila tak, aby při resamplingu původní vývoj amplitudy byl zachován [47]. Tedy, fáze byla násobena koeficientem určujícím chtěný posun tónu (2 pro posun o oktávu nahoru a 0.5 pro posun o oktávu dolů). Příklad vypočtu nové fáze `newPhase` v jazyce C lze vidět v ukázce kódu 4.1. Kromě samotné úpravy fáze odpovídajícím koeficientem `pitchRatio` se provádí i korekce fáze v závislosti na předchozích výpočtech. Proměnná `phase` označuje fázi získanou v daném kroku po aplikaci FFT, `prevPhase`, `prevNewPhase` je `phase`, respektive `newPhase` z minulého kroku a `phaseBin` je konstantní pole hodnot fází jenž odpovídají indexům výsledku FFT. Implementace vycházela ze zdroje [48].

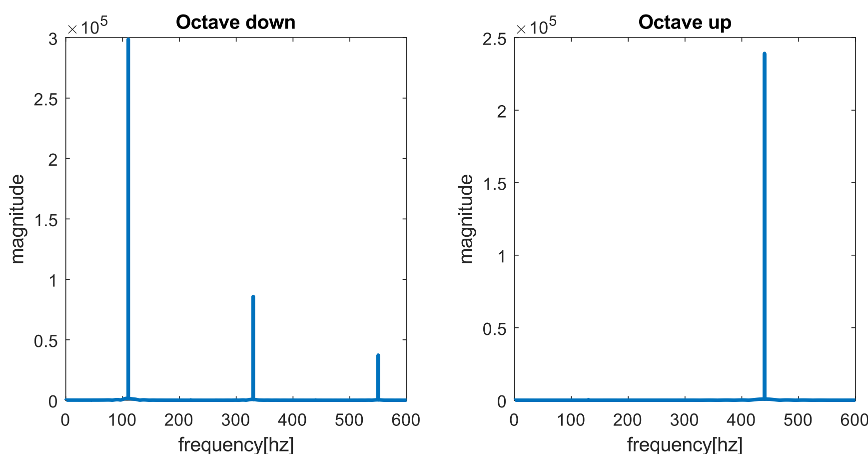
```
if (firstIteration) {

  memcpy(newPhase, phase, sizeof(double) * (FFT_SIZE / 2 + 1));
  firstIteration = 0;

} else {

  for (int i = 0; i < FFT_SIZE / 2 + 1; i++) {
    const double val = prevNewPhase[i] + prevPhaseDiff[i] *
   pitchRatio;
    newPhase[i] = val - round(val / (2 * M_PI)) * 2 * M_PI;
  }

}

for (int i = 0; i < FFT_SIZE / 2 + 1; i++) {
  const double val = phase[i] - prevPhase[i] - phaseBin[i];
  phaseDiff[i] = val - round(val / (2 * M_PI)) * 2 * M_PI +
   phaseBin[i];
}
```

Zdrojový kód 4.1: Phase vocoder, výpočet phase s ohledem na korekci

Za využití upravené fáze a původních magnitud se za pomoci IFFT (viz 1.3.2) resyntézoval upravený signál. Ten se následně za pomoci lineární interpolace převzorkoval buď s 2 krát vyšší frekvenci v případe nižší oktávy, nebo 2 krát nižší v případe oktávy vyšší. Převzorkovaný výsledek se uložil do bufferu a dokud se nevyčerpal, byl následně přičítán k výstupu.

Poznámka 4.3. Phase vocoderem se označuje algoritmus založený na krátkodobé Fourierove transformaci (FFT prováděně postupně v kratkých úsecích), který umožňuje zpracovat signál ve frekvenční doméně rozkladem na sinusoidy a po upravě jejich zpětnou resyntézou. Základem je také průběžná úprava fáze ve snaze zabránit rušení se fáze.[49]

Plugin byl implementován s oknem o délce 1024 vzorků, 2048 bodovým FFT a krokem 64 vzorků. Plugin umožňoval míchat originální signál se signálem posunutým o oktávu, od znění pouze originálu, po znění pouze posunutého. Výstup pluginu, pouze pro posunutý signál, pro sinusovku o 220 Hz lze vidět na obrázku [4.6].



Obrázek 4.6: Octaver, frekvenční spektrum výstupu pro vstup o 220 Hz

4.8 Reverb

Reverbem, nebo reverberaci, se označuje efekt vzniklý součtem odražených zvuků ze stejného zdroje, které díky odrazům od různých povrchů, mají různé zpoždění a energii. Takovýto efekt je znatelný například v prázdné místnosti, či divadelních halách.[50, 51]

Implementace byla založena na Schroeder Reverberatoru [50]. Jedná se o reverberator založený na sekvenčních allpass filtrech a paralelních Feedback Comb Filtrech (Filtry zpožďující signál s se zpětnou vazbou). Schema použité variace je uvedeno na obrázku 4.7. All-pass filtry měly diferenční rovnici 4.2,

$$y[n] = -gx[n] + x[n-N] + gy[n-N] \tag{4.2}$$

kde se hodnoty g a N volily dle doporučení Schroedera [50]

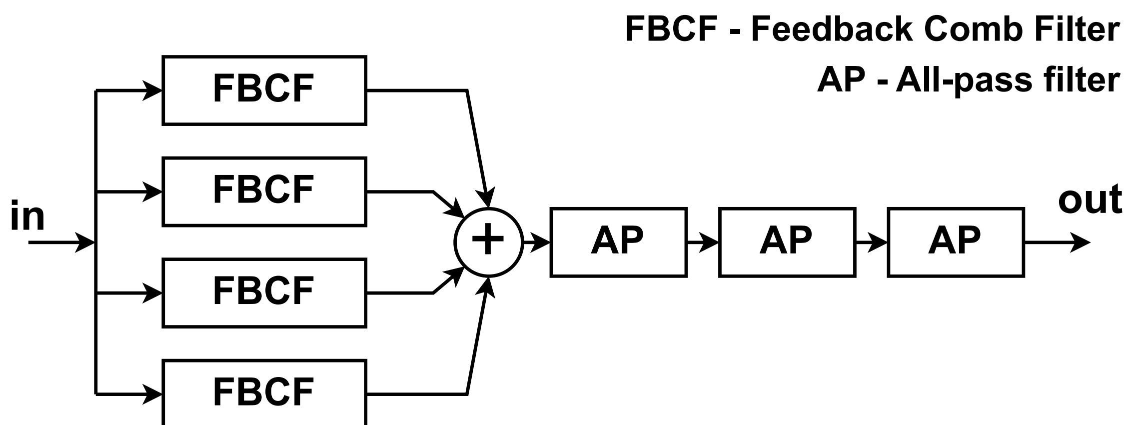
$$g = 0.7$$
$$N = \left\lceil \frac{f_s}{10 \cdot 3^i} \right\rceil \tag{4.3}$$

kde i je celé číslo označující index all-pass filtru od vstupu k výstupu. Tedy, na obrázku [4.7] nevíce levý all-pass filter bude mít index 0, po něm následující 1 a tak dále.

Feedback Comb filtry se dají popsat následující rovnici

$$y[n] = x[n] + y[n - N], N \in \mathbb{N} \tag{4.4}$$

Lze tedy vidět, že se jedna o zpoždění signálu s feedbackem. Velikost prodlevy každého filtru se nechala nastavitelnou v rozsahu 10 ms až 100 ms.



Obrázek 4.7: Reverb

4.9 Biquad Filter

Jedná se o lineární filtr, jehož přenosová funkce se dá vyjádřit jako poměr dvou kvadratických funkcí, viz rovnice 4.5.[52]

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \tag{4.5}$$

Rovnici lze normalizovat dle a_0, pak přenosová funkce bude obsahovat pouze pět koeficientů,

$$H(z) = \frac{\left(\frac{b_0}{a_0}\right) + \left(\frac{b_1}{a_0}\right) z^{-1} + \left(\frac{b_2}{a_0}\right) z^{-2}}{1 + \left(\frac{a_1}{a_0}\right) z^{-1} + \left(\frac{a_2}{a_0}\right) z^{-2}} \tag{4.6}$$

pak výpočet n-tého vzorku může vypadat následovně

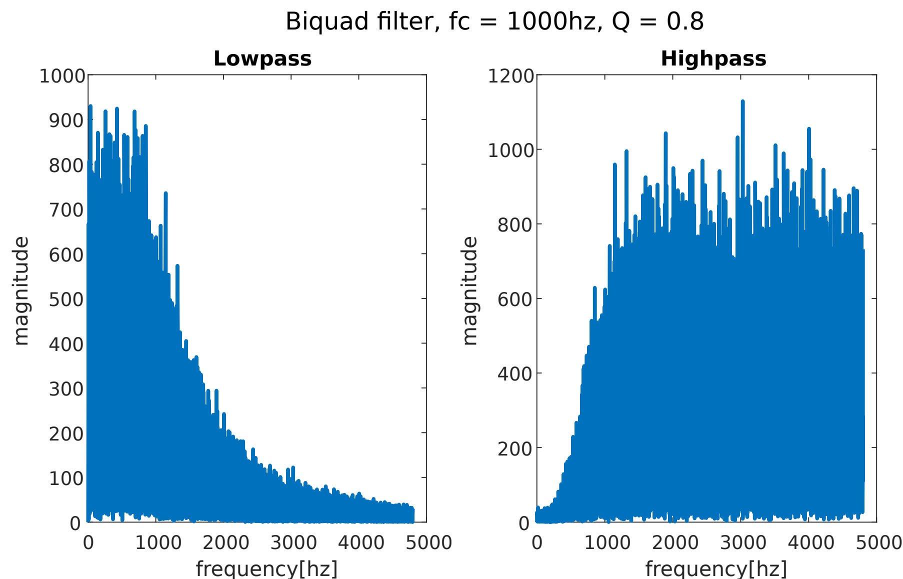
$$y[n] = \left(\frac{b_0}{a_0}\right) x[n] + \left(\frac{b_1}{a_0}\right) x[n-1] + \left(\frac{b_2}{a_0}\right) x[n-2] - \left(\frac{a_1}{a_0}\right) y[n-1] - \left(\frac{a_2}{a_0}\right) y[n-2] \tag{4.7}$$

V závislosti na koeficientech se pak můžou vyjádřit filtry různých typů. Například výpočet koeficientu low-pass filtru může vypadat následovně

```
const double w0 = (2 * M_PI * fc) / (double) sampleRate;
const double a = sin(w0) / (2 * Q);

const double b0 = (1 - cos(w0)) / 2;
const double b1 = 1 - cos(w0);
const double b2 = (1 - cos(w0)) / 2;

const double a0 = 1 + a;
const double a1 = -2 * cos(w0);
const double a2 = 1 - a;
```

Zdrojový kód 4.2: Vypočet koeficientu low pass filtru

kde `fc` je mezní frekvence a `Q` kladná reálná hodnota určující tvar zářezu frekvenčního spektra.



Obrázek 4.8: Biquad filter, odezva na bílý šum

5 Závěr

Výstupem práce je jednoduchý program pro Microsoft Windows umožňující real-time zpracování audia napsaný v jazyce C++. Zpracování audia bylo realizováno za pomoci pluginů ve formě DLL souborů. Ke komunikaci pluginu s programem bylo navrženo rozhraní v jazyce C. Za pomoci rozhraní byly implementovány vybrané audio efekty. Pluginy lze využit jak pro samotné zpracovaní, tak i zobrazení signálu.

Program je relativně lehký. V základu využívá kolem 4 MB RAM a na disku zabírá okolo 800 kB. Prakticky veškerá logika je oddělena od operačního systému. Kromě prostředku ke komunikaci se zvukovými kartami, jejichž samotné implementace jsou také odděleně od logiky programu, program neobsahuje žádné závislosti. Navíc, veškeré vykreslovaní probíhá softwarově. Lze tedy předpokládat, že by program mohl být relativně jednoduše přenesen i na nějaké vestavěné zařízení, například vývojovou desku s ARM čipem. To by mohl byt další směr vývoje aplikace.

Program však není zdaleka odladěn a doplněn o všechny chtěné funkce.

V první řadě by se měla rozšířit podpora různých datových formátů, jak u ASIO, tak i WASAPI a implementace otestovat na řadě různých zvukových kartách. Momentálně program zaručeně funguje se zvukovou kartou, která se používala při vývoji. Ostatní zvukové karty již můžou používat jiné datové formáty. Samozřejmě, samotná podpora mohla být provedena i bez hardwaru, ovšem, nemohla by být ověřena, a tedy se pominula. Program i přes to obsahuje možnost pro komunikaci s velkou řadou zvukových karet za pomoci legacy rozhraní Windows, které bylo implementováno s širší podporou datových formátů a využívá standardní cestu komunikace přes Windows, avšak na úkor latence. V nejhorším případě lze program využívat offline pro souborový vstup a výstup.

Rozmyslet by se měl i typ podnětu programu, prozatím program reaguje jak na podněty uživatele přes události, tak i obsahuje paralelní tick-rate smyčku, která zpracovává připadně periodické jevy (například animaci kurzoru, či vykreslení signálu v čase). V takovémto případě, pokud se něco bude vykreslovat každý tick, veškeré zprávy o vykreslení pocházející z událostí jsou zbytečné. Program může v takovýchto situacích zbytečně přetěžovat procesor. Při reálném použiti, i na relativně nízkých frekvencích (kolem 800 MHz), se však neprojevovali známky nedostačujícího výkonu. Tedy, možná se jedná o paranoidní záležitost, ale i tak by se měla zvážit možnost převodu aplikace pouze na tick-rate smyčku. To by bylo i výhodné kvůli synchronizaci (bylo by jen jedno vlákno, které by mohlo vytvářet podněty pro vykreslení).

Vylepšit by se měl i samotný render. Momentálně program prakticky necachuje data, a až při samotném vykreslení provádí velké množství výpočtů. Kromě toho program moc neřeší oblast vykreslení, a prakticky vždy vykresluje všechny komponenty. Je to hlavně spojené s paralelním tick-rate vláknem, které nemůže napřímo vykreslovat komponenty, ale musí posílat zprávy do hlavního vlákna, které následné provádí vykreslení. Protože prozatím nebylo rozhodnuto, zda paralelní vlákno zůstane, tak nebyl napsán zpravující systém, který by byl schopen zajištění přenosu potřebné informace.

Program by také mohl být obohacen o nové pluginy, které by nesloužily pro samotné zpracování signálu, ale nabízeli nějaké doplňující funkce. Například metronom, nahrávání záznamu do souboru, nebo třeba přehrávaní souboru ve smyčce.

Samotné rozhraní by se mělo obohatit o podporu některých základních prvků jako přepínací tlačítko, nebo posuvníku. Navíc by se mohly přidat i pokročilejší prvky, například menu pro procházení a výběr souborů.

Seznam literatury

[1] WIKIPEDIA. *Digital signal* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://en.wikipedia.org/wiki/Digital_signal.

[2] WIKIPEDIA. *Diskrétní signál* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://cs.wikipedia.org/wiki/Diskr%C3%A9tn%C3%AD_sign%C3%A1l.

[3] WIKIPEDIA. *Nyquist frequency* [online]. 2022 [cit. 2022-04-19]. Dostupné z: https://en.wikipedia.org/wiki/Nyquist_frequency.

[4] WIKIPEDIA. *Discrete Fourier transform* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://en.wikipedia.org/wiki/Discrete_Fourier_transform.

[5] WIKIPEDIA. *Fast Fourier transform* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://en.wikipedia.org/wiki/Fast_Fourier_transform.

[6] AL., Zölzer U. et. *DAFX - Digital Audio Effects*. John Wiley & Sons, 2002. ISBN 0-471-49078-4.

[7] CCRMA. *Transfer Function Analysis* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://ccrma.stanford.edu/~jos/fp/Transfer_Function_Analysis.html#chap:tfd.

[8] WIKIPEDIA. *Z-transform* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://en.wikipedia.org/wiki/Z-transform.

[9] MATHWORLD, Wolfram. *Apodization Function* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://mathworld.wolfram.com/ApodizationFunction.html.

[10] MATHWORKS. *hann* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://www.mathworks.com/help/signal/ref/hann.html.

[11] WIKIPEDIA. *Modulation* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://en.wikipedia.org/wiki/Modulation.

[12] WIKIPEDIA. *Digital audio workstation* [online]. 2022 [cit. 2022-04-19]. Dostupné z: https://en.wikipedia.org/wiki/Digital_audio_workstation.

[13] AUDACITY. *PLUG-INS* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://www.audacityteam.org/about/features/plug-ins/.

[14] AUDACITY. *Nyquist* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://manual.audacityteam.org/man/nyquist.html.

[15] AUDACITY. *ASIO Audio Interface* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://manual.audacityteam.org/man/asio_audio_interface.html.

[16] AUDACITY. *Real-time preview of effects* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://manual.audacityteam.org/man/real_time_preview_of_effects.html.

[17] REAPER. *About* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://www.reaper.fm/about.php.
[18] REAPER. *Purchase* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://www.reaper.fm/purchase.php.
[19] STEINBERG. *ELICENSER* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://www.steinberg.net/elicenser/.
[20] STEINBERG. *Cubase* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://www.steinberg.net/cubase/.
[21] HOELSCHER, Joey. *Cubase vs Pro Tools: Which DAW Is the Right DAW For YOU?* [Online]. 2022 [cit. 2022-04-08]. Dostupné z: https://lancetingey.com/cubase-vs-pro-tools/.
[22] AVID. *PRO TOOLS* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://www.avid.com/pro-tools.
[23] AVID. *Which Pro Tools audio recording software is right for you?* [Online]. 2022 [cit. 2022-04-08]. Dostupné z: https://www.avid.com/audio-recording-software.
[24] APPLE. *GarageBand for iOS* [online]. 2022 [cit. 2022-04-23]. Dostupné z: https://www.apple.com/ios/garageband/.
[25] APPLE. *GarageBand* [online]. 2022 [cit. 2022-04-23]. Dostupné z: https://apps.apple.com/app/garageband/id408709785?ls=1&v0=www-us-ios-garageband-app-garageband.
[26] NICHOLAS. *Plugin Formats Explained (VST, AU, AAX, etc)* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://support.pluginboutique.com/hc/en-gb/articles/360007431078-Plugin-Formats-Explained-VST-AU-AAX-etc-.
[27] STEINBERG. *OUR TECHNOLOGIES* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://www.steinberg.net/technology/.
[28] APPLE. *Audio Unit* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://developer.apple.com/documentation/audiounit.
[29] INSTRUMENTS, NATIVE. *Guitar Rig 6 Pro* [online]. 2022 [cit. 2022-04-23]. Dostupné z: https://www.native-instruments.com/en/products/komplete/guitar/guitar-rig-6-pro/.
[30] INSTRUMENTS, NATIVE. *Guitar Rig 6 Player* [online]. 2022 [cit. 2022-04-23]. Dostupné z: https://www.native-instruments.com/en/products/komplete/guitar/guitar-rig-6-player/.
[31] INSTRUMENTS, NATIVE. *GUITAR RIG COMPARISON CHART* [online]. 2022 [cit. 2022-04-23]. Dostupné z: https://www.native-instruments.com/en/products/komplete/guitar/guitar-rig-6-pro/comparison-chart/.
[32] INSTRUMENTS, NATIVE. *SYSTEM REQUIREMENTS* [online]. 2022 [cit. 2022-04-23]. Dostupné z: https://www.native-instruments.com/en/products/komplete/guitar/guitar-rig-6-pro/specifications/.
[33] MICROSOFT. *Low Latency Audio* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/low-latency-audio.
[34] GRABIT, Yvan. *ASIO ? What is it?* [Online]. 2022 [cit. 2022-04-08]. Dostupné z: https://forums.steinberg.net/t/asio-what-is-it/201552.

[35] WIKIPEDIA. *Audio Stream Input/Output* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://en.wikipedia.org/wiki/Audio_Stream_Input/Output.

[36] *ASIO 2.3 Audio Streaming Input Output Development Kit*. 2019. Dostupné také z: https://www.steinberg.net/developers/. Release#3.

[37] ASIO4ALL. *asio4all* [online]. 2022 [cit. 2022-04-08]. Dostupné z: https://www.asio4all.org.

[38] EWIUSB. *ASIO4ALL - ESSENTIAL SOFTWARE THAT SHOULDN'T NEED TO EXIST!* [Online]. 2022 [cit. 2022-04-08]. Dostupné z: https://ewiusb.com/review_ASIO4ALL.

[39] WIKIPEDIA. *Distortion* [online]. 2022 [cit. 2022-04-25]. Dostupné z: https://en.wikipedia.org/wiki/Distortion.

[40] WIKIPEDIA. *Distortion (music)* [online]. 2022 [cit. 2022-04-25]. Dostupné z: https://en.wikipedia.org/wiki/Distortion_(music).

[41] CCRMA. *Hard Clipping* [online]. 2022 [cit. 2022-04-25]. Dostupné z: https://ccrma.stanford.edu/~jos/pasp/Hard_Clipping.html.

[42] CCRMA. *Soft Clipping* [online]. 2022 [cit. 2022-04-25]. Dostupné z: https://ccrma.stanford.edu/~jos/pasp/Soft_Clipping.html.

[43] ELECTROSMASH. *Boss DS-1 Distortion Analysis* [online]. 2022 [cit. 2022-04-25]. Dostupné z: https://www.electrosmash.com/boss-ds1-analysis.

[44] WIKIPEDIA. *Phaser (effect)* [online]. 2022 [cit. 2022-04-24]. Dostupné z: https://en.wikipedia.org/wiki/Phaser_(effect).

[45] WIKIPEDIA. *Vibrato* [online]. 2022 [cit. 2022-04-19]. Dostupné z: https://en.wikipedia.org/wiki/Vibrato.

[46] WIKIPEDIA. *Octave effect* [online]. 2022 [cit. 2022-04-25]. Dostupné z: https://en.wikipedia.org/wiki/Octave_effect.

[47] LAROCHE, Jean; DOLSON, Mark. Phase-vocoder: About this phasiness business. In: *Proceedings of 1997 Workshop on Applications of Signal Processing to Audio and Acoustics*. 1997, 4–pp.

[48] LIM, Kyung Ae; INFORMATICS, Music. An Open-Source Phase Vocoder with Some Novel Visualizations. *Indiana University Bloomington, USA*. 2007.

[49] DANNENBERG, Roger B. *Phase Vocoder Tutorial* [online]. 2022 [cit. 2022-05-01]. Dostupné z: https://www.cs.cmu.edu/~music/nyquist/extensions/pvoc/phasevocoder.html.

[50] CCRMA. *Schroeder Reverberators* [online]. 2022 [cit. 2022-04-30]. Dostupné z: https://ccrma.stanford.edu/~jos/pasp/Schroeder_Reverberators.html.

[51] TOMA, Norbert; TOPA, Marina; SZOPOS, Erwin. Aspects of reverberation algorithms. In: *International Symposium on Signals, Circuits and Systems, 2005. ISSCS 2005.* 2005, sv. 2, s. 577–580.

[52] W3C. *Audio EQ Cookbook* [online]. 2022 [cit. 2022-05-01]. Dostupné z: https://www.w3.org/TR/audio-eq-cookbook.

A Příloha

Obsah přiloženého USB flash disku:

- Samotná práce v elektronické podobě ve formátu pdf
- Zdrojové soubory
- SDK pro implementaci pluginů
- Dokumentace rozhraní pro tvorbu pluginů obsahující tutoriál a API referenci ve formátu HTML
- Zdrojové kódy programu MATLAB obsahující testy a případné implementace některých audio efektů
- Testovací vstupní a odpovídající výstupní soubory použité pro testování každého efektu