VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

HUDEBNÍ EFEKT "GLITCH MACHINE"

SOUND EFFECT "GLITCH MACHINE"

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MATEJ CIMMERMAN**
AUTHOR

VEDOUCÍ PRÁCE **prof. Dr. Ing. JAN ČERNOCKÝ**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

148963

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Student:	**Cimmerman Matej**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Hudební efekt "glitch machine"**
Kategorie:	Zpracování signálů
Akademický rok:	2022/23

Zadání:

1. Seznamte se s principy digitálních hudebních efektů a softwarovými nástroji pro jejich tvorbu a integraci.
2. Proveďte rešerši stávajících řešení efektu "glitch machine".
3. Implementujte tento efekt ve zvoleném programovacím jazyce, průběžně kontrolujte kvalitu výstupu.
4. Vytvořte metodiku uživatelského hodnocení, najděte skupinu testerů, hodnocení proveďte a vyhodnoťte.
5. Na základě testů navrhněte a implementujte vylepšení či rozšíření efektu.
6. Navrhněte postup další práce (integrace do jednoho z hudebních SW, atd.).
7. Vytvořte o Vaší práci krátké video.

Literatura:

- dle doporučení vedoucího.

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Černocký Jan, prof. Dr. Ing.**
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	31.10.2022

Abstrakt

Práca sa venuje problematike softvérových hudobných efektov a ich implementácii. Rieši návrh a implementáciu programu Glitch Machine, ktorý patrí medzi hudobné multiefektové programy a demonštruje simultánne použitie rôznych hudobných efektov pre úpravu zvukových signálov.

Výsledkom je program pre systém Windows, ktorý využíva spoluprácu rôznych efektov, a to konkrétne filter, reverb, distortion, extractor, reverz, stutter, shifter, pitch resample a gain. Umožňuje rozsiahlu a detailnú úpravu zvukovej stopy a vyššiu mieru manipulácie so zvukmi, než pri samostatných hudobných efektoch. Ďalšími súčasťami programu sú aj zobrazenia zvukovej vlny a frekvenčného spektra zvukových signálov a možnosť načítania a ukladania efektových presetov.

Program bol testovaný vlastnými testami výkonu a využívania pamäte a tiež užívateľským testovaním kvality a grafického rozhrania.

Abstract

This thesis addresses software music effects and their implementation. It shows design and implementation of the Glitch Machine, which belongs to the music multieffect programs and demonstrates the simultaneous use of various music effects to process audio signals.

The result is a program for Windows OS, which utilizes cooperation of various effects, namely a filter, reverb, distortion, extractor, reverz, stutter, shifter, pitch resample and gain. It allows extensive and detailed editing of an audio track and a higher extent of manipulation with sounds, than with single music effects. Other components of the program include visualisations of the waveform and frequency spectrum of audio signals and an option for loading and saving effect presets.

The program was tested in tests of the performance and memory usage and also with user testing of quality and graphical interface.

Klíčová slova

hudba, zvuk, efekt, reverb, delay, flanger, distortion, softclip, hardclip, rectify, extractor, reverz, stutter, shifter, JUCE, frekvencia, vlna, modulácia, filter, Fourierova transformácia, časová doména, frekvenčná doména

Keywords

music, sound, effect, reverb, delay, flanger, distortion, softclip, hardclip, rectify, extractor, reverz, stutter, shifter, JUCE, frequency, waveform, modulation, filter, Fourier transform, time domain, frequency domain

Citace

CIMMERMAN, Matej. *Hudební efekt "glitch machine"*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Dr. Ing. Jan Černocký

Hudební efekt "glitch machine"

Prohlášení

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Doc. Dr. Ing. Jana Černockého. Ďalšie informácie mi poskytol Ing. Jan Kaňka z Audified s.r.o. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Matej Cimmerman
6. května 2023

Poděkování

Chcel by som sa poďakovať môjmu vedúcemu tejto práce, Janovi Černockému, za konzultácie, pripomienky a rady, ktoré boli nesmierne dôležité pre dokončenie tejto práce. Ďalej by som chcel poďakovať môjmu externému konzultantovi Janovi Kaňkovi za nápady a vôbec za tému, ktorej sa táto práca venuje. Ďalšia vďaka patrí testerom môjho programu, a to Andrejovi Srnovi, Danielovi Paulovičovi a Marekovi Mesárošovi. Na záver by som chcel poďakovať aj členom rodiny, ktorí boli mojou oporou pri dokončovaní tejto práce.

Obsah

Slovník použitých výrazov

AP	all-pass (filter)
BP	band-pass (filter)
Buffer	dátová štruktúra obsahujúca vzorky digitálneho zvukového signálu
DAW	Digital Audio Workstation - softvér na tvorbu hudobných skladieb schopný využívať pluginy.
Delay	oneskorenie (oneskorovacia linka)
Distortion	skreslenie (tvaru zvukovej vlny)
FFT	fast Fourier transform
Framework	štrukturovaný súbor knižníc pre podporu vývoja aplikácie (vývojová platforma)
HP	high-pass (filter)
LF	low frequency
LFO	low frequency oscillator
Plugin	aplikácia vo verzii zásuvného modulu pre použitie v DAW
Reverb	dozvuk (dozvukový efekt)

Kapitola 1

Úvod

Hudobné efekty sú dôležitou súčasťou tvorby a úpravy hudobných skladieb už od počiatku používania elektronických zariadení v hudobnom priemysle. S príchodom softvéru pre digitálnu tvorbu hudby tieto efekty prešli zmenou z hardvérových riešení na softvérové. Programy pre hudobné efekty existujú v rôznych podobách od jednoduchých samostatných efektov, až po multiefekty schopné využívať väčšie množstvo efektov pre úpravu zvuku.

Glitch Machine patrí medzi tieto multiefektové programy a poskytuje užívateľovi možnosť komplexnej úpravy zvukovej stopy. Je určený hlavne pre žánre patriace do modernej elektronickej hudby a jeho cieľom je umožniť manipuláciu so zvukom od jemných úprav až po úplné "znetvorenie".

Táto bakalárska práca sa zaoberá kompletným návrhom a implementáciou programu hudobného efektu Glitch Machine. Glitch Machine je implementovaný vo forme samostatného programu v C++ pomocou framework-u JUCE. Obsahuje efekty rôzneho charakteru a účelu, pričom si užívateľ môže vybrať, ktoré efekty chce použiť a v akom poradí ich chce použiť pre väčšiu flexibilitu a prispôsobiteľnosť pri úprave zvukovej stopy. Grafické rozhranie efektu je vytvorené s ohľadom na grafické rozhrania existujúcich programov, aby bol užívateľ schopný používať program intuitívne bez potreby zdĺhavej orientácie v rozhraní. Pre ďalšie uľahčenie používania program obsahuje niekoľko užívateľských preset-ov, ktoré obsahujú nastavenia parametrov jednotlivých efektov. Tieto presety sa dajú aplikovať okamžite, alebo použiť ako inšpirácia pre vlastné nastavenie efektov.

V druhej kapitole sa nachádza prieskum existujúcich riešení glitch efektov a ich stručná charakteristika a subjektívne hodnotenie z pohľadu užívateľa. Tretia kapitola obsahuje krátky teoretický popis niektorých základných prvkov využívaných v práci. Štvrtá kapitola popisuje návrh samotných efektov. Piata kapitola popisuje prostriedky využité pri implementácii. Šiesta kapitola popisuje implementáciu jednotlivých prvkov programu. Siedma kapitola rozoberá návrh a implementáciu grafického užívateľského rozhrania. Ôsma kapitola sa zaoberá testovaním programu, spracovaním testov a aplikovaním vylepšení programu. Posledná deviata kapitola obsahuje návrh možných rozšírení programu a rekapituláciu práce.

Kapitola 2

Prieskum existujúcich riešení

2.1 Glitch Machine by Stagecraft

Cena: 60$.

GM od Starecraft obsahuje grain efekt, ktorý rozdeľuje signál na množstvo menších úsekov a na tieto úseky následne aplikuje ďalšie operácie. Medzi operáciami môže byť napr. zmena výšky, oneskorenie, posunutie do ľavého/ pravého kanála (panning), atď.

Tento multiefekt umožňuje aj podrobnejšie ovládanie zmeny výšky po aplikácii grain efektu a to napr. miera zmeny výšky, alebo množstvo vzoriek, ktorým sa výška zmení. Rozhranie multiefektu obsahuje grafické znázornenie spracovaného signálu v reálnom čase a tiež vykreslenie frekvenčného spektra s filtrom.

Obrázek 2.1: GM efekt od Stagecraft. Prevzaté z [23]

2.2 Fracture by GlitchMachines

Cena: zadarmo.

Fracture od GlitchMachines obsahuje buffer, delay a filter s nastaviteľným typom, kvalitou a frekvenciou. Buffer efekt načítava spracovaný signál do dočasnej pamäte a tento úsek signálu v pamäti následne opakovane prehráva. Buffer má nastaviteľnú veľkosť načítavaného signálu a počet opakovaní prehrávania. Delay je jednoduchá oneskorovacia linka so základnými nastaveniami. Tento efekt je tu na doplnenie Buffer efektu, pretože z umeleckého

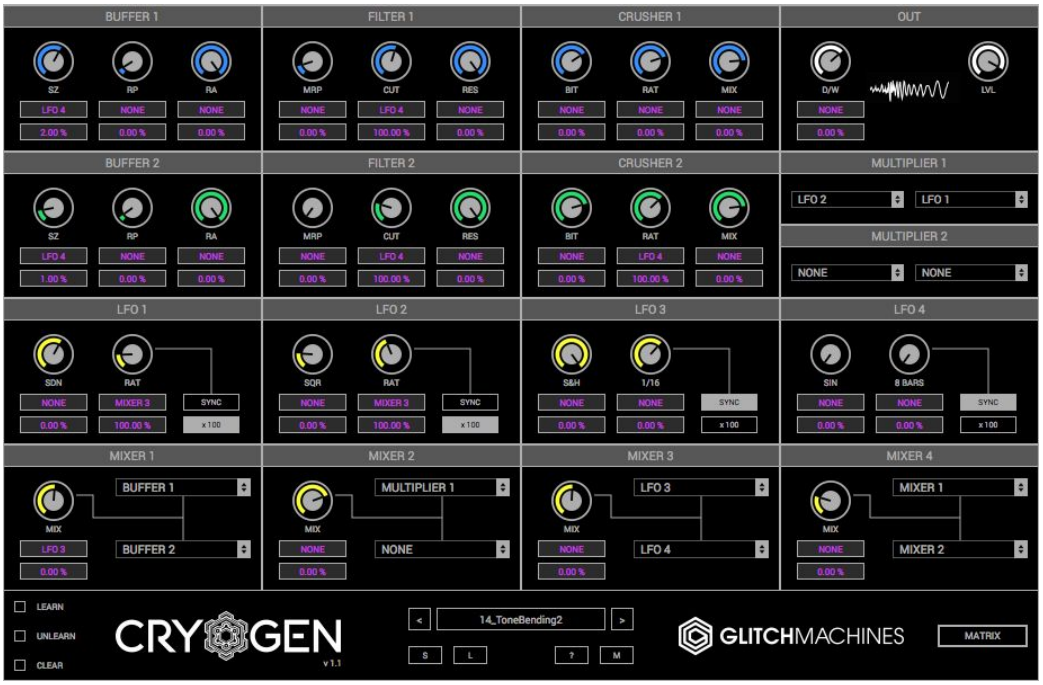
hľadiska sa dá vhodne spojiť s buffer efektom. Buffer efekt ukladá časti zvukového signálu v doplnkovom bufferi a následne ich opakovane prehráva spolu s pôvodným signálom.

Obrázek 2.2: GM efekt Fracture od GlitchMachines. Prevzaté z [8]

2.3 Cryogen by GlitchMachines

Cena: 59$.

Posledným skúmaným multiefektom je Cryogen, ktorého jadro obsahuje 2 nastaviteľné buffer efekty, 2 filtre a 2 bitcrush distortion efekty, ktoré skresľujú tvar zvukovej vlny. Okrem týchto obsahuje aj 4 LFO oscilátory, ktoré sa dajú aplikovať na ľubovoľný nastaviteľný parameter hociktorého z efektov.



Obrázek 2.3: GM efekt Cryogen od GlitchMachines. Prevzaté z [6]

2.4 Polygon by GlitchMachines

Cena: 79$.

Polygon nie je typický Glitch Machine multiefekt, ale je to sampler, ktorý dokáže generovať zvukový signál na základe sample-based syntézy. To znamená, že patrí medzi generátory zvukov a nie medzi efekty, čiže ho nie je možné aplikovať na zvukový signál ako efekt.

Na tomto efekte je zaujímavé spojenie rôznych efektov, medzi nimi aj efektov využívaných pre Glitch Machine, a zvukového syntetizátora upraveného pre generovanie špecifických glitch efektov.

Obrázek 2.4: GM efekt Polygon od GlitchMachines. Prevzaté z [20]

2.5 Získané poznatky

Na základe prieskumu sme získali lepšiu predstavu o tom, aké sú momentálne možnosti medzi Glitch Machine multiefektami na trhu a aké prvky by mohol obsahovať riešený Glitch Machine.

V skúmaných efektoch bol braný ohľad na konkrétne efekty, prídavné funkcie, kvalitu zvuku a aj na realizáciu grafického rozhrania. Kvalita zvuku bola na vysokej úrovni vo všetkých efektoch. Čo sa efektov týka, tak samozrejme programy Polygon a Cryogen majú najväčší výber efektov, avšak každý zo skúmaných programov je rovnako dobre využiteľný. Grafické rozhranie bolo práveže v jednoduchších efektov prehľadnejšie, ale žiadny z efektov nemá rozhranie, ktoré by po určitom čase bolo aj naďalej nepoužiteľné.

Implementovaný program by mohol fungovať na princípe, ktorý využíva jednoduché a všeobecne známe efekty v spolupráci s glitch efektami a umožňuje aplikáciu týchto efektov v ľubovoľnej kombinácii. Následne by mohol obsahovať prídavné ovládacie a zobrazovacie

prvky, ako napr. LFO oscilátory v programe Cryogen, alebo vizualizér zvukovej vlny ako v Polygon-e.

Na záver, grafické rozhranie programu (hlavne ovládacie prvky) by malo byť implementované podobne, ako vo všetkých existujúcich programoch na trhu, keďže tento štýl je používaný už od prvých softvérových hudobných efektov a užívatelia sú s ním oboznámení. Rozhranie by tiež malo dávať užívateľovi grafickú odozvu toho, čo sa vykonáva so spracovaným zvukom (zobrazenie zvukovej vlny, spektrograf, atď).

Kapitola 3

Teória

3.1 Digitálny zvukový signál

Zvuk je v digitálnych softvérových efektoch spracovávaný ako digitálny zvukový signál. Takýto digitálny signál má viacero špecifických vlastností, ktoré sú dôležité pre úpravu signálu v rámci softvérových efektov.

Vzorkovacia frekvencia

V prvom rade sa od analógového spojitého signálu odlišuje tým, že je zložený z konečného počtu vzoriek, ktoré majú určité hodnoty, pretože digitálne systémy sú schopné pracovať len s digitálnymi dátami.

Počet vzoriek signálu je určený vzorkovacou frekvenciou F_s, ktorá udáva počet vzoriek v 1s signálu. Najpoužívanejšou hodnotou je 44100Hz.

Vzorkovacia frekvencia udáva kvalitu signálu, ktorá je priamo úmerná jej hodnote, ale zároveň aj maximálnu možnú frekvenciu zvuku, ktorú je možné vyjadriť týmto signálom. Maximálna možná frekvencia sa riadi tzv. vzorkovacou teorémou (*Nyquist-Shannon theorem*):

$$f = \frac{F_s}{2} \tag{3.1}$$

Čo znamená, že napr. pri vzorkovacej frekvencii 44100Hz je maximálna zreprodukovateľná frekvencia 22050Hz [24].

Bitová hĺbka

Jednotlivé vzorky sú vždy uložené pod určitým dátovým typom, ktorého veľkosť udáva aj veľkosť vzorky. Veľkosť priamo súvisí s bitovou hĺbkou b signálu. Bitová hĺbka označuje počet možných hodnôt, ktoré môže nadobúdať hodnota signálu. Počet možných hodnôt je definovaný nasledovne:

$$2^b \tag{3.2}$$

Počet bitov b je daný dátovým typom. Bežne používanými dátovými typmi sú napr. `32bit float`, `24bit int` a `16bit int`. Bitová hĺbka tiež ovplyvňuje kvalitu a spolu so

vzorkovacou frekvenciou sa používa pre určenie bitovej rýchlosti zvukového signálu, čo je hlavný parameter používaný pre určenie kvality signálu [25].

3.2 Rýchla Fourierova transformácia

FFT umožňuje prevod signálu v časovej doméne do jeho frekvenčnej domény. Frekvenčné spektrum signálu je rozdelené na jeho frekvenčné zložky a ich rozmiestnenie v rámci spektra je dané vzorkovacou frekvenciou signálu.

Dôležitými parametrami FFT z pohľadu implementácie sú:

- Vzorkovacia frekvencia
- Veľkosť bloku (počet vzoriek)

Maximálna frekvencia zložiek je daná teorémou uvedenou v 3.1, čiže je rovná polovici vzorkovacej frekvencie.

Veľkosť bloku určuje frekvenciu spracovania signálu. Od nej závisí rozlíšenie frekvenčného spektra a rýchlosť obnovovania zobrazeného spektra. Platí, že čím väčšia je veľkosť bloku, tým vyššie je rozlíšenie, ale zároveň tým nižšia je rýchlosť obnovy. V softvérových aplikáciách je často nutné nájsť kompromis medzi kvalitou výsledného zobrazenia a frekvenciou obnovovania. Bežne používané hodnoty veľkosti bloku sú napr. 2^{10}, 2^{11} alebo 2^{12} a v našom programe je táto hodnota nastavená na 2^{12}.

FFT je vhodné hlavne pre periodické signály a signály, ktoré majú v rámci bloku celé periódy. V digitálnych signáloch s konečnou dĺžkou sú pri rozdelení signálu do blokov jednotlivé periódy prerušované, čo vedie ku vzniku náhlych zmien hodnôt signálu. Tieto zmeny produkujú tzv. “spectral leakage” a spôsobujú vznik ďalších harmonických frekvencií (pretože náhla zmena hodnoty má podobnú frekvenčnú charakteristiku ako jednoduchá obdĺžniková vlna) [19].

Pre zníženie nežiaducich vlastností sa používa okienková funkcia. Táto funkcia vyhladzuje hodnoty na okrajoch každého bloku a zmierňuje vznik nežiaducich frekvencií vo výslednom spektre. Najčastejšie používanými funkciami sú napr. Hannova a Hammingova okienková funkcia.

Kapitola 4

Návrh jednotlivých efektov

4.1 Delay

Delay slúži na simuláciu oneskorenia zvukového signálu. V základnej podobe je využiteľný len v nízkej miere, ale používa sa ako základný prvok mnohých ďalších efektov. Funguje na princípe duplikovania vstupného signálu a jeho oneskorenia o určitý počet vzoriek.

Môžme si vyjadriť rovnicu popisujúcu základnú formu delay-u:

$$w[n] = x[n] + g \cdot w[n - N] \tag{4.1}$$

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

n je index konkrétnej vzorky v signáli

$x[n]$ značí pôvodný vstupný signál

$w[n]$ je signál, ktorý sa posiela na spracovanie

$w[n - N]$ je signál, ktorý prešiel oneskorením a posiela sa späť na spracovanie

$y[n]$ je konečný výstupný signál

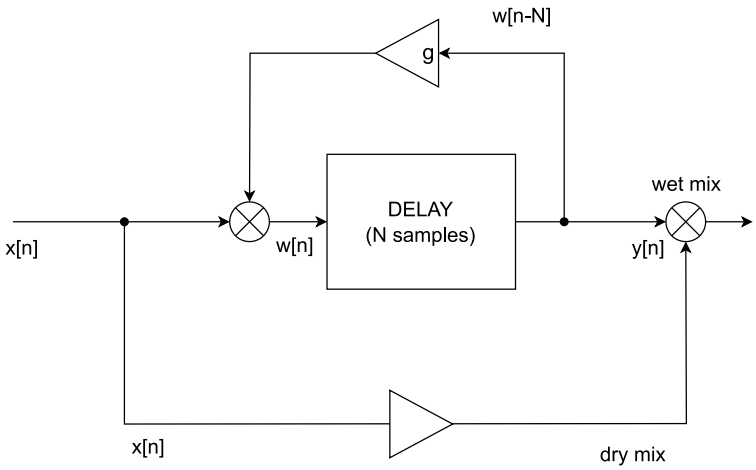
g je koeficient, ktorým sa násobí spracovaný signál

N je ľubovoľný počet vzoriek, o ktorý sa signál oneskoruje

Vyjadrený delay je rozšírenou verziou základného, ktorý iba oneskoruje signál. Obsahuje aj možnosť zmeny pomeru spracovaného a nespracovaného signálu a tiež ovládanie vplyvu efektu cez parameter g.

Popis delay-u je dôležitý pre porozumenie ďalších efektov, ale znázornená základná verzia tohto efektu vykonáva oneskorenie signálu len raz. V programe by mal byť tento efekt implementovaný tak, aby vykonával oneskorenie viackrát podľa nastavenia.

Delay je znázornený na obrázku 4.1



Obrázek 4.1: Schéma oneskorovacej linky.

4.2 Reverb

Reverb je efekt simulujúci ozvenu určitého priestoru. Ozvena v reálnom svete vzniká odrážaním zvukových vĺn z určitého zdroja od povrchov rôznych objektov v okolí. Keď tieto odrazené zvukové vlny cestujú do ucha poslucháča, tak ľudský mozog ich spracováva a vníma ako ozvenu. Ozvena má množstvo vlastností, na základe ktorých je človek schopný vďaka nej do určitej miery vnímať priestor, v ktorom sa nachádza. Tieto vlastnosti sú ovplyvňované napr. hlasitosťou zdroja, veľkosťou zdroja, vzdialenosťou a rozložením objektov v priestore a fyzikálnymi vlastnosťami materiálov daných objektov.

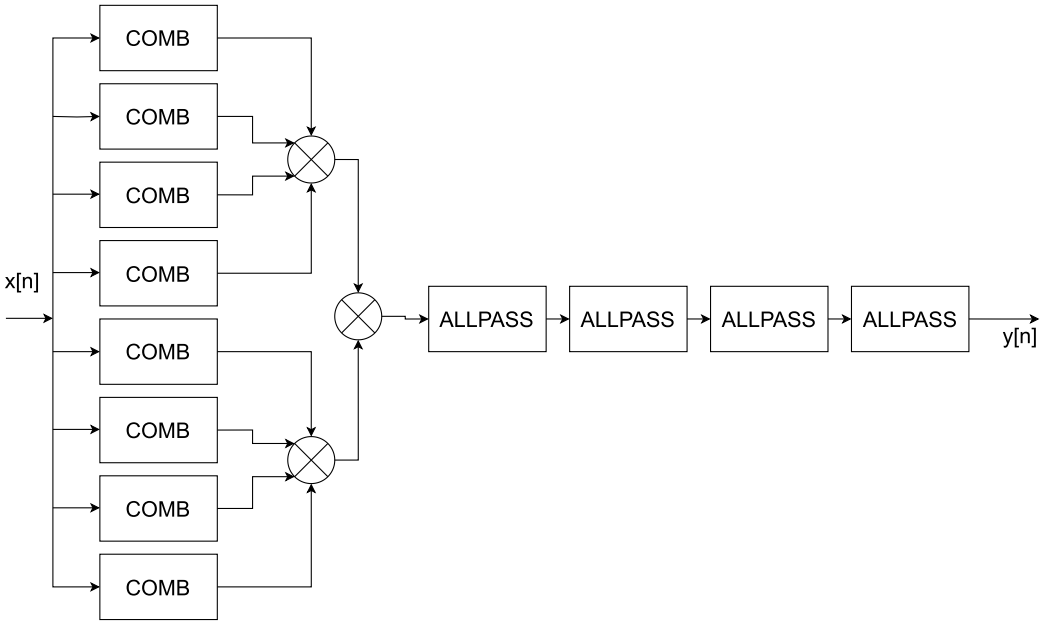
Pri návrhu reverbu musíme vedieť, čo vlastne ozvena robí. Pri ozvene existuje určitý zdrojový zvukový signál, ktorý sa pri každom odraze od objektu zoslabí. Pri ozvene toto odrážanie signálu môže a aj väčšinou prebieha opakovane. To znamená, že sa daný signál kopíruje a filtruje v závislosti od povrchu objektov, ale vždy s nižšou hodnotou. Toto kopírovanie je podobné, ako pri oneskorovacej linke. Má niekoľko odlišností - signál sa kopíruje viackrát (je použitých viacero liniek) a signál sa pri kopírovaní zoslabí [7].

4.2.1 Schroederov reverb

Existuje viacero typov reverbov, ale pre projekt bol vybraný tzv. Využíva viacero oneskorovacích liniek a ľubovoľné množstvo allpass filtrov [21], ako je zobrazené na obrázku 4.2. Schroederov reverb má niekoľko vlastností, ktoré je možné prispôsobiť:

- Vplyv efektu na spracovaný signál
- Trvanie reverberácie. Dlhšie trvanie efektu vytvára dojem väčšieho priestoru.
- Mieru prepúšťania vysokých frekvencií.
- Šírku efektu v priestorovom spektre.

Konkrétny reverb vybraný pre projekt je implementovaný z knižnice JUCE s názvom `Reverb.h`. Tento reverb je postavený na základe open-source reverb-u s názvom `FreeVerb`. Využíva 8 comb filtrov, z ktorých každý je zložený z jednoduchej oneskorovacej linky. Jej výstup je upravený LP filtrom, vynásobený určitým koeficientom a následne sčítaný so



Obrázek 4.2: Schéma Schroederovho reverbu. [10]

vstupom danej linky. Výstupy každej linky sa zároveň sčítavajú (ako vidno na schéme) a sú posielané do série allpass filtrov, pričom tento reverb využíva 4 filtre [10].

Oneskorovacie linky v comb filtroch slúžia na vytvorenie samotného reverb efektu. Koeficient g (uvedený na obrázku 4.1), ktorým je výstup linky násobený má hodnotu v rozmedzí $(0, 1)$ a pri každom násobení zníži hodnotu výstupu z linky. Tento koeficient slúži na nastavenie dĺžky reverbu (priamo úmerná koeficientu) od nulovej do teoreticky nekonečnej dĺžky.

All-pass filtre následne slúžia na vyhladenie a zjemnenie výsledného reverbu (bez nich by boli počuteľné jednotlivé oneskorenia signálu a efekt by znel skôr ako viacnásobný delay) a použitím viac, než jedného filtra je dosiahnuteľný lepší výsledný efekt.

4.3 Chorus

Chorus je realizovateľný ako kombinácia viacerých oneskorovacích liniek s pridaným nízkofrekvenčným oscilátorom. Oneskorovacie linky majú variabilné oneskorenie signálu, ktorého zmeny sú vykonávané pomocou LFO oscilátora. LFO generuje signál s určitým tvarom vlny (väčšinou je používaná jednoduchá sínusoida) a následne je posielaný do oneskorovacej linky, kde je pomocou daného signálu modulovaná hodnota oneskorenia signálu. Kvôli modulácii časového oneskorenia efekt patrí medzi tzv. efekty s časovou moduláciou (taktiež nazývanou aj fázová modulácia) [4]. Na obrázku 4.3 je uvedený príklad efektu s tromi oneskorovacími linkami (každá predstavuje celú linku uvedenú v 4.1) a tromi LFO oscilátormi.

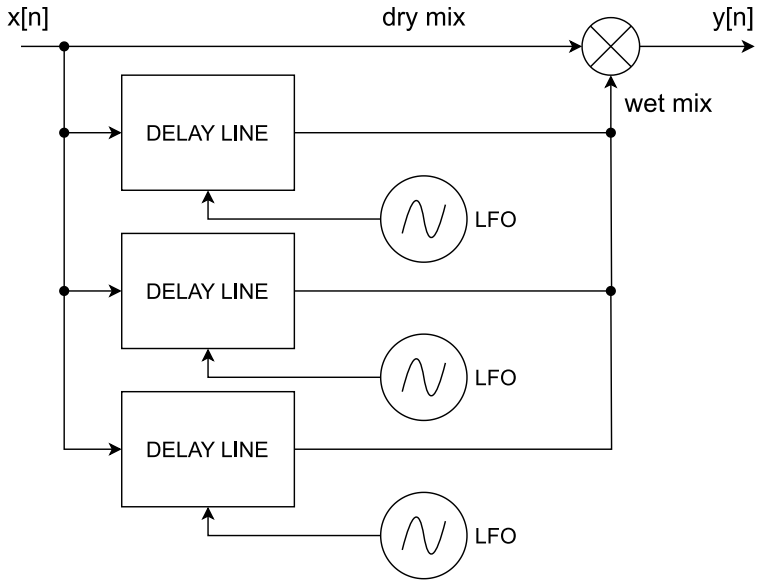
Chorus je v projekte implementovaný na základe knižnice z JUCE. Tento efekt obsahuje niekoľko nastaviteľných vlastností:

- Frekvenciu LFO oscilátora pre moduláciu signálu
- Hodnotu zvukovej vlny LFO oscilátora

- Mieru znižovania hodnoty signálu v oneskorovacích linkách (parameter g uvedený v Delay efekte 4.1)
- Dĺžku oneskorenia jednotlivých liniek

Chorus slúži na vytvorenie "chorálneho" efektu, ktorý znie ako reprodukcia pôvodného zvuku z viacerých zvukových výstupov, z ktorých je každý mierne odlišný. Toto je dôvod, kvôli ktorému je v efekte využité vyššie množstvo oneskorovacích liniek (simulácia viacerých výstupov) a kvôli ktorému sú oneskorenia každej linky zvlášť modulované cez LFO.

V skutočnosti pri chorálnom speve je hlas každého účastníka mierne odlišný nielen v čase, ale aj vo výške. Chorus tento efekt však dosahuje iba cez časovú moduláciu (nie aj cez frekvenčnú moduláciu), keďže časový posun viacerých kópií signálu v tomto prípade vytvára aj mierne zmeny vo výške zvuku - čomu napomáha aj to, akým spôsobom je tento efekt vnímaný ľudským sluchom.



Obrázek 4.3: Schéma Chorus efektu.

4.4 Distortion

Distortion slúži na skresľovanie vstupného signálu. Efekt pracuje na princípe úpravy tvaru zvukovej vlny, pričom existuje viacero možností, ako tvar upraviť. Reálne sa zvykne využívať napr. pre zvýraznenie zvuku, vytváranie ostrejších zvukov, alebo naopak pre utlmenie a zjemnenie zvukov.

V tomto programe by mal byť tento efekt využitý pre extenzívnejšiu úpravu zvukového signálu, a to až do miery, kedy daný zvuk prestáva byť všeobecne počúvateľný.

Môžme uvažovať nad viacerými typmi distortion efektov, a to:

4.4.1 Hard Clipping

Hard clipping je jednoduchá forma skreslenia, ktorá funguje nasledovne:

$$y[n] = \begin{cases} t & \text{pre } x[n] > t \\ -t & \text{pre } x[n] < t \\ x[n] & inak \end{cases} \tag{4.3}$$

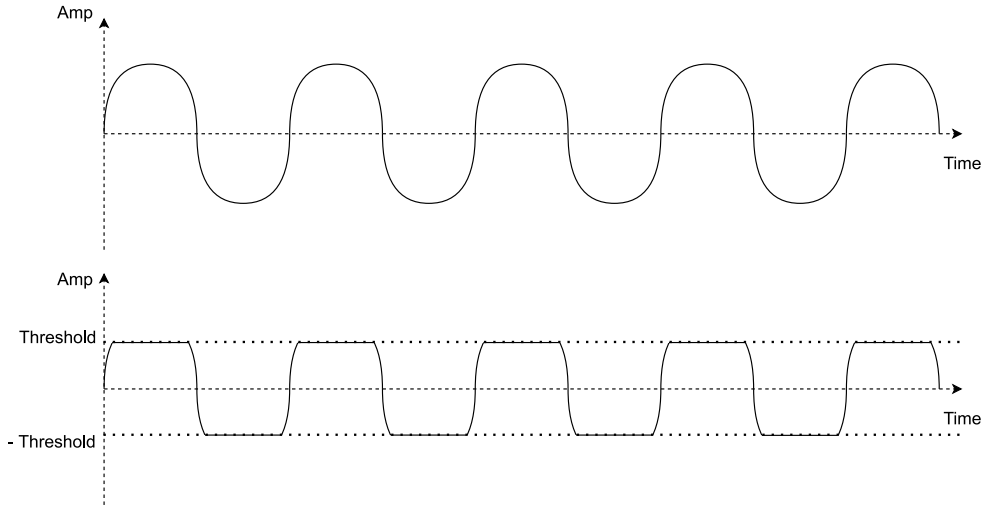
kde:

$x[n]$ je hodnota vstupného signálu

$y[n]$ je hodnota výstupného signálu

$(-)t$, tzv. *threshold*, je hodnota maximálnej amplitúdy signálu

Vstupný signál je orezávaný na základe nastavenej hodnoty *threshold*, čiže ak amplitúda presiahne túto hodnotu, tak bude nastavená práve na túto hodnotu. Hodnota zvukovej vlny je týmto obmedzená na rozsah od *-threshold* do *+threshold*.



Obrázek 4.4: Zvuková vlna pred / po úprave.

Zvuk sa pri úprave týmto efektom svojím znením približuje k zneniu obdĺžnikovej zvukovej vlny, čo je možné vidieť aj na obrázku 4.4, pretože aj tvar zvukovej vlny sa takisto približuje na obdĺžnikovú. Tento efekt je možné využiť najmä dvomi najvyužívanejšími spôsobmi.

Prvé využitie má pri úprave zvukov, ktorých zvukové vlny sú jednoduchšie, než obdĺžniková vlna, a to hlavne z hľadiska množstva harmonických frekvencií, z ktorých sa skladajú. V tomto prípade čiastočná zmena tvaru vlny vedie k pridaniu ďalších harmonických frekvencií do pôvodného zvuku. Využíva sa to pri úprave basov a basových liniek a to hlavne na nízkofrekvenčné basy (iným názvom sub / sub-bass), pretože tento typ basov je väčšinou zložený z jednoduchej sínusovej vlny. Sínusové zvukové vlny obsahujú len jednu základnú frekvenciu (a žiadne ďalšie harmonické frekvencie) a preto bez dodatočnej úpravy nie sú dobre zreprodukovateľné na zvukových systémoch, ktoré nemajú dostatočný frekvenčný rozsah. Napr. sínusový sub-bass s frekvenciou 50 Hz nie je možné prehrať na systéme s rozsahom od 100 Hz. Hardclip (okrem iných) umožňuje pridať do sínusovej vlny ďalšie harmonické frekvencie, ktoré sa nachádzajú vyššie než základná frekvencia, a tým zvýrazniť zvukovú vlnu a umožniť reprodukciu na horších zvukových systémoch.

4.4.2 Soft Clipping

Soft clipping je menej agresívna forma skreslenia, než hard-clipping. Najzákladnejšou používanou metódou je úprava signálu na základe nasledujúcej funkcie:

$$y[n] = \begin{cases} -\frac{2}{3} & \text{pre } x[n] < -1 \\ x[n] - \frac{x[n]^3}{3} & \text{pre } -1 \leq x[n] \leq 1 \\ \frac{2}{3} & \text{pre } x[n] > 1 \end{cases} \tag{4.4}$$

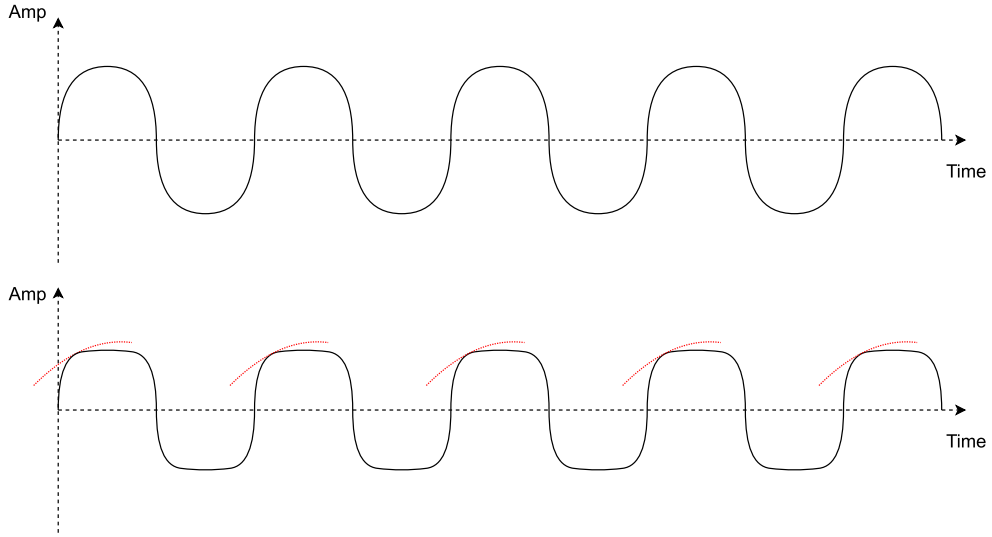
kde:

$x[n]$ je hodnota vstupného signálu

$y[n]$ je hodnota výstupného signálu

Soft clipping je využiteľný pre jemnejšiu úpravu (hlavne z umeleckého pohľadu) zvukového signálu, než pri hard-clippingu. Zvukovú vlna sa od úpravy pomocou hard-clipping efektu líši hlavne tým, že tvar vlny nie je upravovaný výrazným spôsobom pomocou tvrdého obmedzenia hodnoty, ale pomocou určitej funkcie. Najčastejšie je používaná nelineárna kubická funkcia definovaná v rovnici 4.4 [22].

V uvedenej funkcii je možné z rovnice vybrať koeficient, ktorým sa násobí kubická hodnota amplitúdy (v tomto prípade $\frac{1}{3}$) a tento koeficient je možné využiť pre úpravu miery efektu. Pri znížení koeficientu dochádza k zníženiu úrovne efektu, čo sa dá vidieť aj na skutočnosti, ak je koeficient znížený na hodnotu 0, tak z nelineárnej kubickej funkcie sa stáva lineárna funkcia, ktorá zároveň žiadnym spôsobom neovplyvňuje vstupný signál.

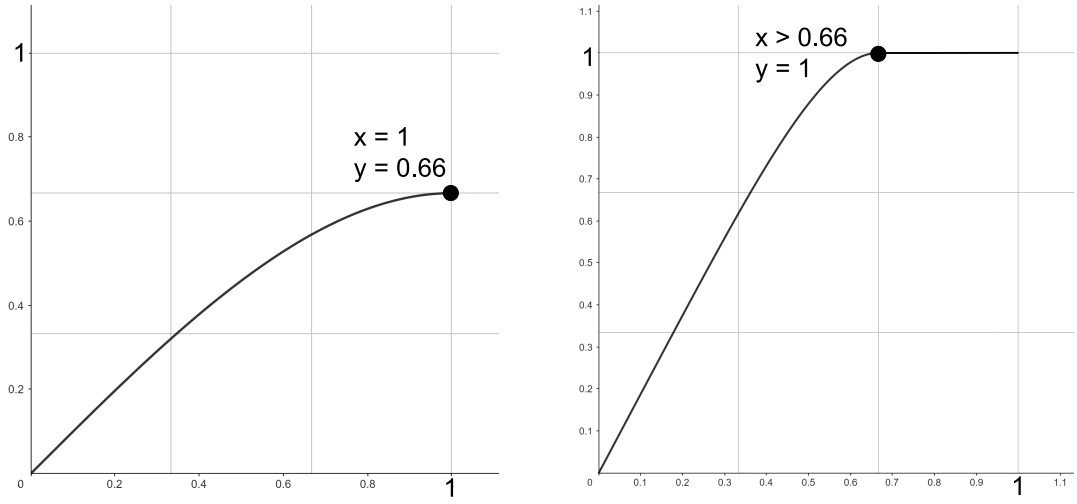


Obrázek 4.5: Zvuková vlna pred / po úprave.

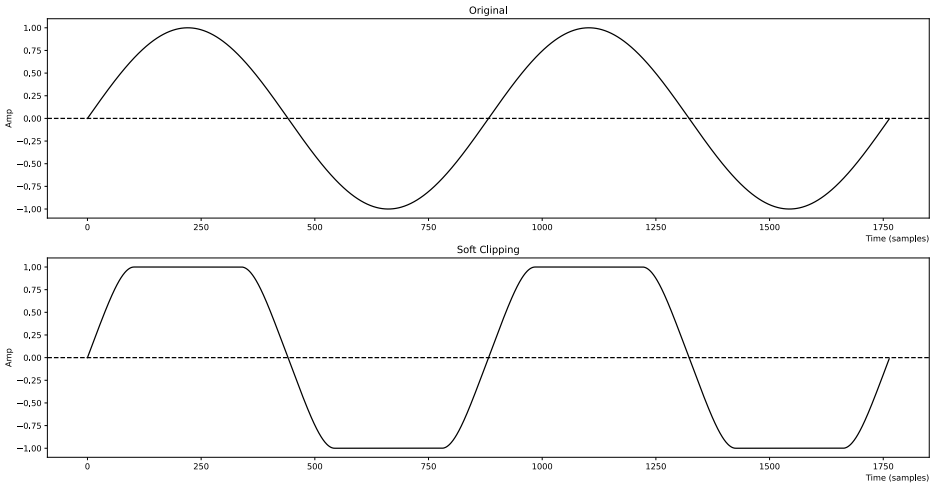
Soft-clipping implementovaný v programe však využíva odlišnú funkciu. Funkcia použitá v našom programe má tvar upravený tak, aby úprava zvukového signálu bola výraznejšia, než v pôvodnej verzii:

$$y[n] = \begin{cases} -1 & \text{pre } x[n] < -0.66 \\ 1.88079 \cdot (A_i - A_i^{4.93917}) & \text{pre } -0.66 \leq x[n] \leq 0.66 \\ 1 & \text{pre } x[n] > 0.66 \end{cases} \tag{4.5}$$

Od pôvodnej verzie sa odlišuje hlavne výraznejším zosilnením vzoriek s nižšou hodnotou a tiež obmedzením maximálnej hodnoty vzoriek na 1 alebo -1, čiže na maximálnu hodnotu vzorky. Vo svojej podstate plní funkciu zvýraznenia a zároveň limitovania zvukového signálu. Rozdiel v zmene vstupných hodnôt na výstupné je vidieť na obrázku 4.6 a rozdiel v tvare zvukovej vlny na obrázku 4.7.



Obrázek 4.6: Pôvodný a vlastný softclip efekt.



Obrázek 4.7: Zvuková vlna pred / po úprave cez vlastný softclip.

4.4.3 Rectify

Rectify existuje v dvoch formách, tzv. full-vawe a half-vawe rectify. Full-vawe rectify mení hodnoty všetkých vzoriek s negatívnou hodnotou na pozitívne hodnoty. To znamená, že full-vawe rectify sa dá popísať funkciou, ktorá mení hodnoty vzoriek na absolútnu hodnotu [9].

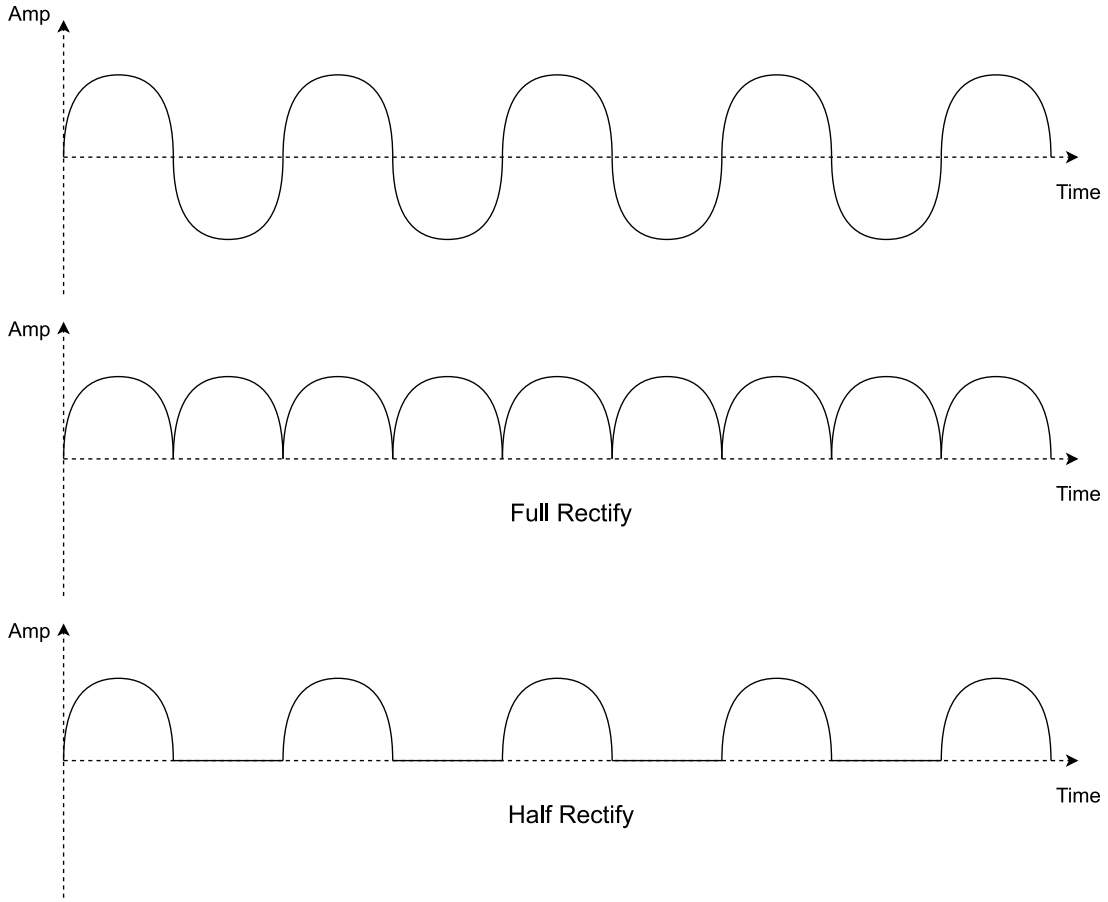
$$y[n] = \begin{cases} |x[n]| & \text{pre } x[n] < 0 \\ x[n] & inak \end{cases} \tag{4.6}$$

Half-vawe rectify mení hodnoty všetkých vzoriek s negatívnou hodnotou na 0 [11]. Half-vawe rectify sa dá popísať nasledovnou funkciou:

$$y[n] = \begin{cases} 0 & \text{pre } x[n] < 0 \\ x[n] & inak \end{cases} \tag{4.7}$$

Rectify efekty kvôli svojmu špecifickému spôsobu úpravy zvukovej vlny spôsobujú zmeny v harmonických frekvenciách spracovaného signálu. Harmonické frekvencie signálu sú znásobované a ich hodnoty sú párnymi násobkami pôvodných frekvencií. Reálne to znamená väčšie množstvo frekvencií vo vyššej časti spektra, než v ktorej sa nachádzal pôvodný signál. Táto zmena je využiteľná pre zvýšenie "jasnosti" zvuku pri zvukoch, ktorým táto vlastnosť chýba. Tento efekt však nie je vhodný pre zvuky, ktoré obsahujú príliš veľa harmonických frekvencií, pretože výsledok môže byť menej žiaduci, než pri jednoduchých zvukoch.

Na obrázku 4.8 vidieť úpravu zvukovej vlny pomocou obidvoch efektov.

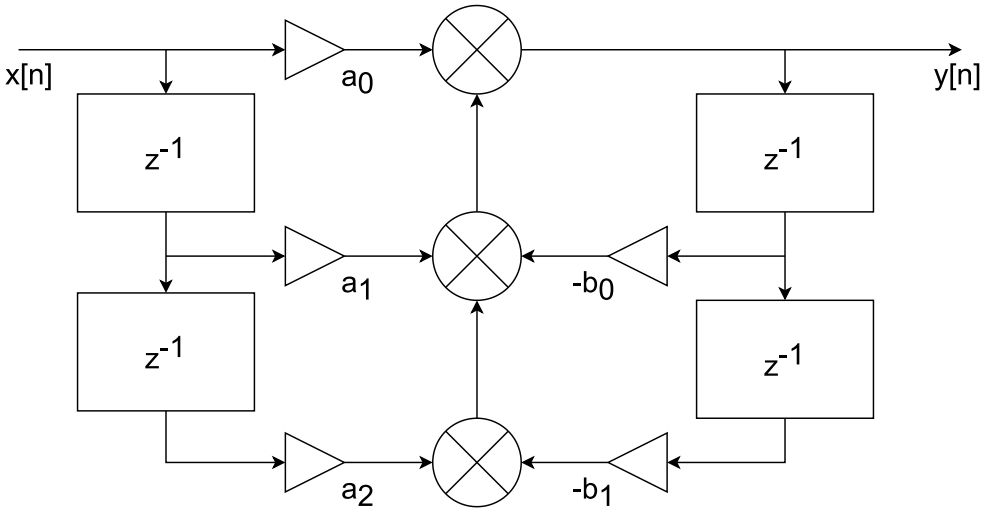


Obrázek 4.8: Zvuková vlna pred / po úprave.

4.5 Filter

Súčasťou multiefektových programov sú vo väčšine prípadov aj niektoré filtre. Filter upravuje zvukový signál vo frekvenčnej doméne, pričom väčšinou slúži na odstránenie určitého frekvenčného pásma.

Spomedzi viacerých druhov filtrov bol zvolený tzv. Biquad (alebo biquadratic) filter. Biquad filter je rekurzívny lineárny filter (využíva predchádzajúce vzorky) s nekonečnou impulznou odozvou (IIR). Výhodou biquad filtra je možnosť využiť jeden filter na implementáciu viacerých druhov filtrov. V tomto programe je využitý na implementáciu low-pass, high-pass a band-pass filtra [1].



Obrázek 4.9: Schéma Biquad filtra.

Biquad filter vykonáva filtrovanie signálu na základe určitých koeficientov a_n a b_n (vidno na obrázku 4.9), ktoré sú vypočítané nasledovne:

Low-pass

a_0	a_1	a_2	b_1	b_2
K^2N	$2a_0$	a_0	$2(K^2-1)N$	$(1K/Q+K^2)N$

High-pass

a_0	a_1	a_2	b_1	b_2
N	$-2a_0$	a_0	$2(K^2-1)N$	$(1-K/Q+K^2)N$

Band-pass

a_0	a_1	a_2	b_1	b_2
$(K/Q)N$	0	$(-K/Q)N$	$2(K^2-1)N$	$(1-K/Q+K^2)N$

Tabulka 4.1: Koeficienty pre biquad filtre. Prevzaté z [2]

Pomocné koeficienty K a N sú vypočítané zvlášť pred výpočtom uvedených koeficientov a_n a b_n:

$$K = \tan(\pi(f_c/F_s)) \qquad N = \frac{1}{1 + \frac{K}{Q} + K^2} \tag{4.8}$$

kde:

f_c je medzná frekvencia filtra, na ktorej sa začína znižovať hodnota signálu

F_s je vzorkovacia frekvencia signálu

Q je kvalita filtra označujúca strmosť krivky

Okrem LP, HP a BP filtrov môže byť biquad filter využitý aj na iné, bežne používané filtre, a to napr.

- Peak - zvyšuje amplitúdu určitého frekvenčného pásma okolo vybranej frekvencie
- Notch - znižuje amplitúdu určitého pásma okolo vybranej frekvencie
- Low-shelf - zvyšuje amplitúdu frekvencií nižších, než vybraná frekvencia
- High-shelf - zvyšuje amplitúdu frekvencií vyšších, než vybraná frekvencia

4.5.1 Low-pass - dolná priepusť

Low-pass (ďalej LP) filter prepúšťa len nízke frekvencie signálu.

Prepúšťané frekvencie je možné nastaviť pomocou parametra `Cutoff`. Tento parameter reprezentuje medznú frekvenciu f_c označujúcu bod vo frekvenčnom spektre signálu, v ktorom sa začína znižovať hodnota signálu.

Ďalším parametrom k dispozícii je Q, ktorý predstavuje kvalitu daného filtra. Tento parameter ovplyvňuje to, ako prudko je signál za frekvenciou f_c filtrovaný, ale zároveň vytvára v bode tejto frekvencie rezonanciu. Rezonancia zvyšuje hodnotu signálu v bode f_c a jeho okolí je pri používaní filtra je očakávateľná a z hľadiska hudobnej produkcie je využiteľná v zvukovom dizajne.

Na obrázku 4.10 vidieť frekvenčnú odozvu biquad LP filtra s centrálnou frekvenciou o hodnote 10000 Hz, s parametrom Q nastaveným na hodnotu 1.0 a so vzorkovacou frekvenciou 44100 Hz.

4.5.2 High-pass - horná priepusť

High-pass (HP) filter prepúšťa len vysoké frekvencie signálu. Má rovnaké parametre ako low-pass filter, len s rozdielom potláčania frekvencií nižších, než f_c a prepúšťania frekvencií vyšších, než f_c.

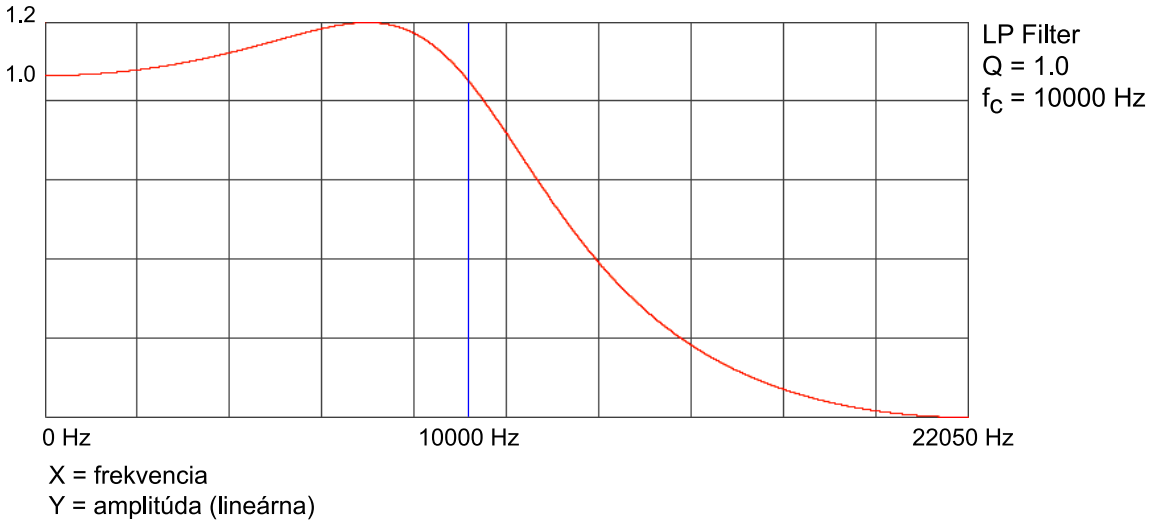
Na obrázku 4.11 vidieť frekvenčnú odozvu biquad HP filtra s $f_c = 10000$ Hz, $Q = 1.0$ a vzorkovacou frekvenciou 44100 Hz.

4.5.3 Band-pass - pásmová priepusť

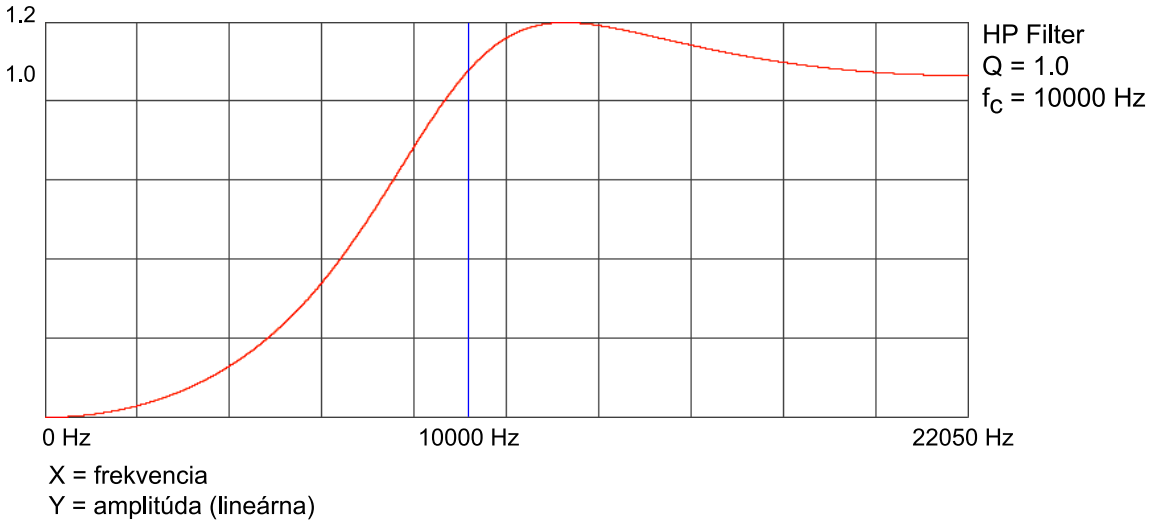
Band-pass (BP) filter prepúšťa stredné frekvencie signálu. Rovnako, ako pri predchádzajúcich filtroch používa parametre f_c a Q, ale využíva ich odlišným spôsobom.

Parameter f_c označuje stredovú frekvenciu okolo ktorej sa vykonáva filtrovanie. To znamená, že je zachované frekvenčné pásmo signálu s určitou šírkou. Šírka prepusteného pásma je určená parametrom Q, ktorý v tomto prípade slúži pre zúženie / rozšírenie tohto pásma. Parameter kvality filtra v tomto prípade nezvyšuje hodnotu signálu v bode f_c, ako v LP a HP filtroch, ale určuje len strmosť krivky frekvenčnej odozvy.

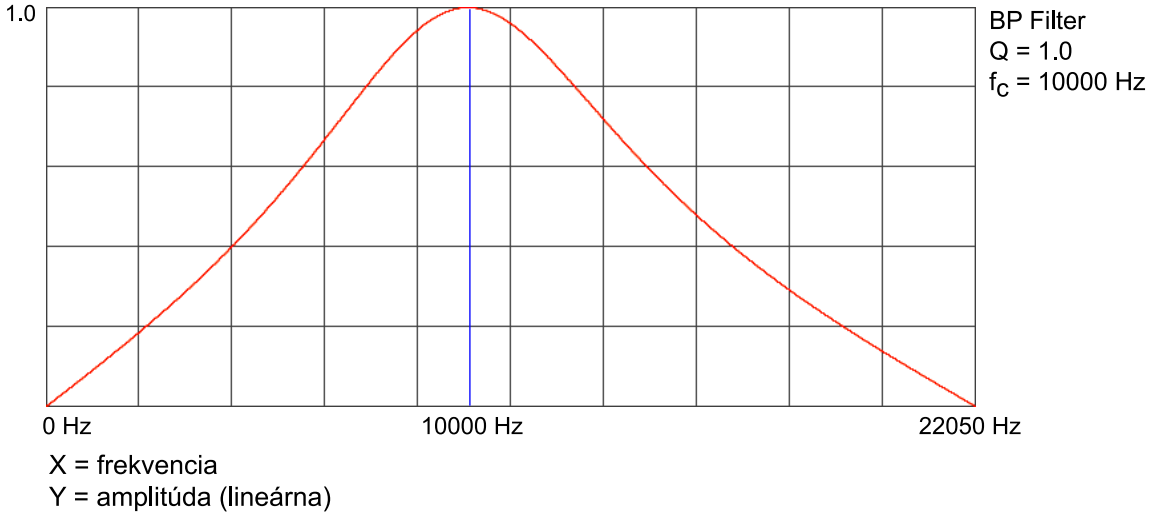
Na obrázku 4.12 vidieť frekvenčnú odozvu biquad BP filtra s $f_c = 10000$ Hz, $Q = 1.0$ a vzorkovacou frekvenciou 44100 Hz.



Obrázek 4.10: Frekvenčná odozva Biquad LPF. Prevzaté z [3].



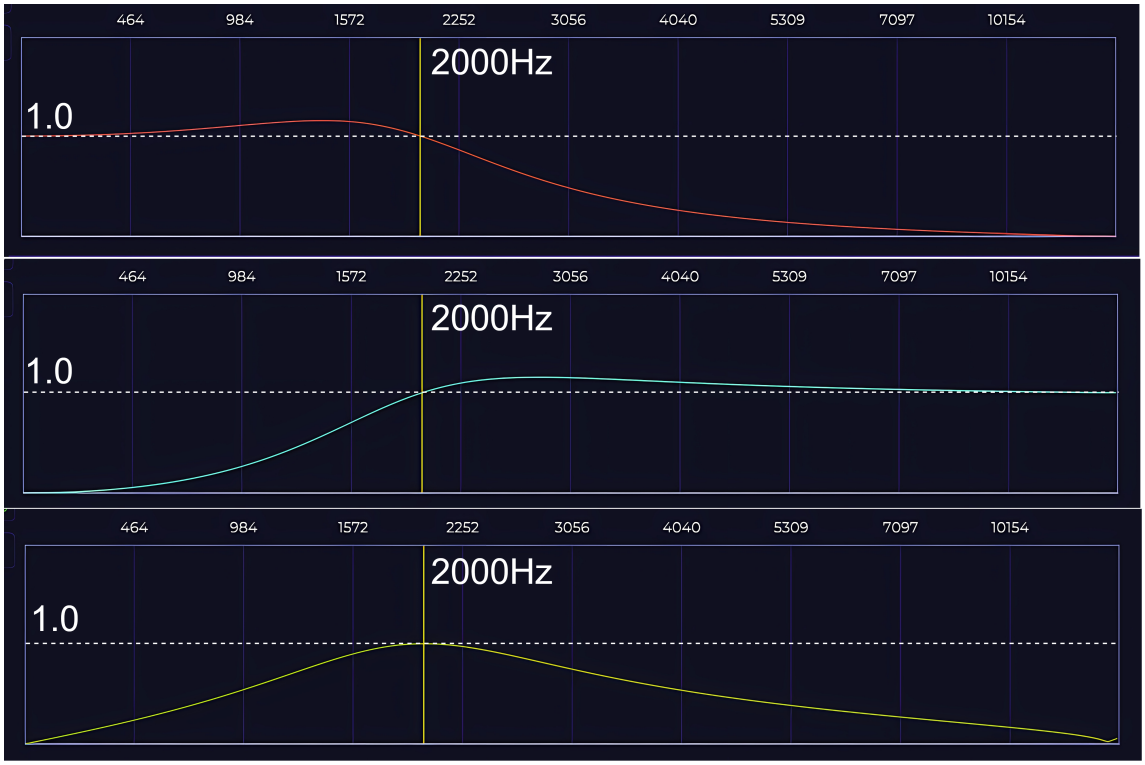
Obrázek 4.11: Frekvenčná odozva Biquad HPF. Prevzaté z [3].



Obrázek 4.12: Frekvenčná odozva Biquad BPF. Prevzaté z [3].

4.5.4 Skutočné odozvy filtra

Obrázky 4.10, 4.11, 4.12 obsahujú teoretické frekvenčné odozvy vypočítané z koeficientov na základe uvedených parametrov f_c, F_s a Q. Filter implementovaný v našom programe je doplnený o indikáciu frekvenčnej odozvy, ktorú je možné použiť pre overenie predpokladanej frekvenčnej odozvy. Skutočná frekvenčná odozva nášho filtra pre LP, HP a BP filter je uvedená na obrázku 4.13.



Obrázek 4.13: Skutočná frekvenčná odozva Biquad filtra.

4.6 Pitch

Pitch je efekt, ktorý slúži na zmenu frekvencie spracovávaného signálu. Zmena frekvencie je možná v násobkoch od 0.1 do 4.0, kde 0.1 je 10-krát nižšia frekvencia a 4.0 je 4-krát vyššia frekvencia.

Algoritmy, ktoré sa používajú pre zmenu frekvencie sú všeobecne rozdelené na 2 typy:

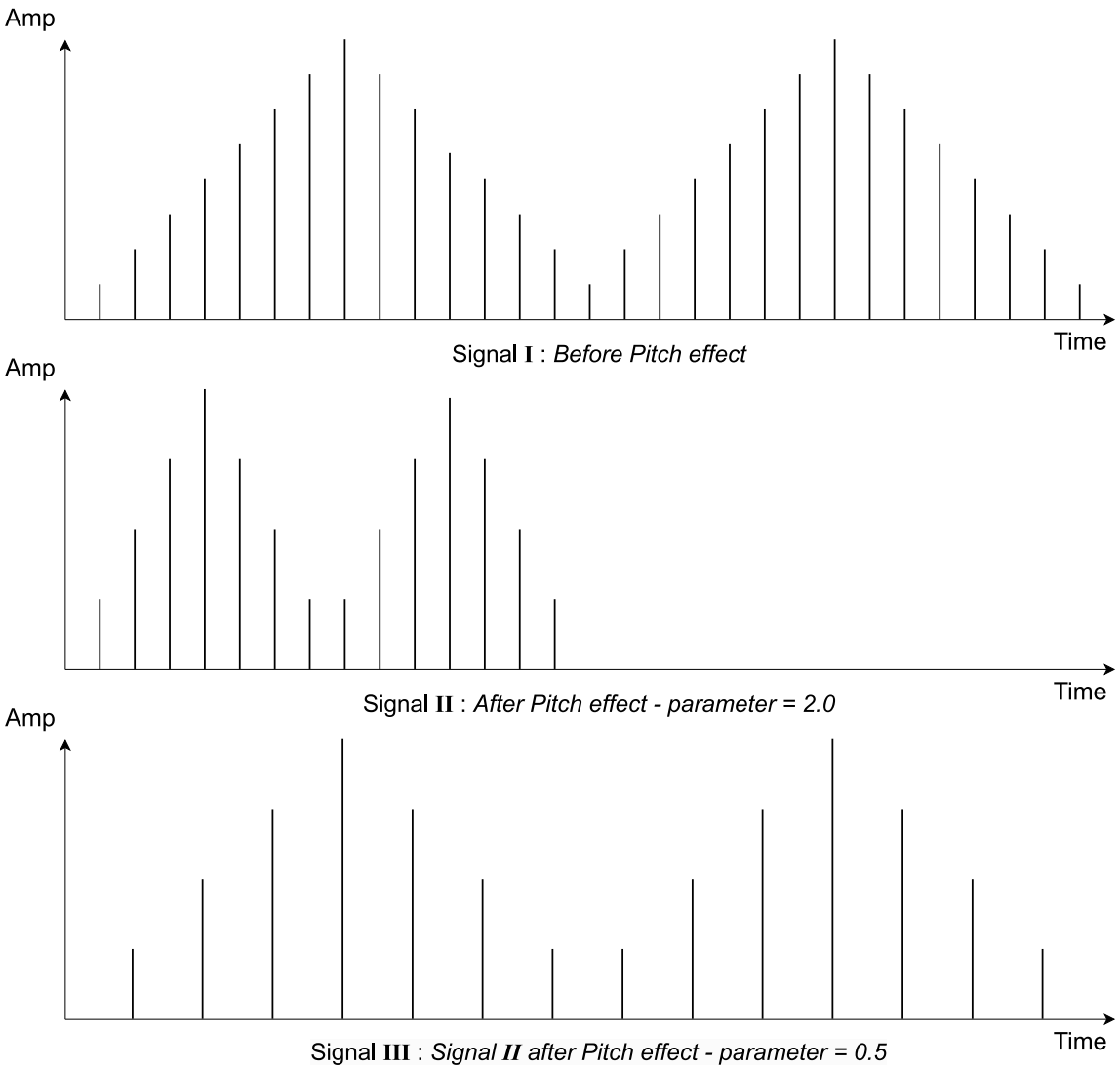
- algoritmy s prevzorkovaním - so zmenou frekvencie sa mení aj dĺžka signálu
- algoritmy bez prevzorkovania - so zmenou frekvencie nedochádza ku zmene dĺžky signálu

Algoritmus vybraný pre tento efekt pracuje na princípe prevzorkovania signálu a pozostáva z dvoch procesov - jeden proces vykonáva zvyšovanie frekvencie a druhý proces vykonáva znižovanie frekvencie. To, že ktorý z dvoch procesov je aktívny závisí od parametra tohto efektu. Ak je parameter v intervale <0.1, 1.0), tak je aktívny proces pre znižovanie frekvencie a v intervale (1.0, 4.0> je aktívny druhý proces.

Pri znižovaní frekvencie je potrebné zväčšiť buffer so signálom a následne rozmiestniť pôvodné vzorky rovnomerne podľa novej dĺžky buffer-u.

Pri zvyšovaní frekvencie je naopak buffer skrátený a následne sú vzorky rozmiestnené podľa novej dĺžky. Pri tomto procese síce dochádza ku strate určitých vzoriek, keďže niektoré vzorky sú nahradené inými, avšak strate určitých vzoriek nie je možné pri skrátení signálu zabrániť .

Výsledok zvýšenia a zníženia frekvencie je znázornený na obrázku 4.14.



Obrázek 4.14: Zvuková vlna pred / po úprave cez Pitch efekt.

4.7 Glitch efekty

4.7.1 Extractor

Extractor je jeden z vlastných efektov, ktorý je navrhnutý pre vytváranie glitch efektu v upravovanej zvukovej stope.

Efekt pracuje na princípe odstraňovania určitých úsekov vzoriek signálu, čím poškodzuje zvuk a vytvára u poslucháča dojem prerušovania zvuku podobný prehrávaniu poškodeného audio média. Na obrázku 4.16 vidieť odstraňovanie úsekov signálu.

Extractor má dve verzie. Obidve verzie majú niekoľko parametrov:

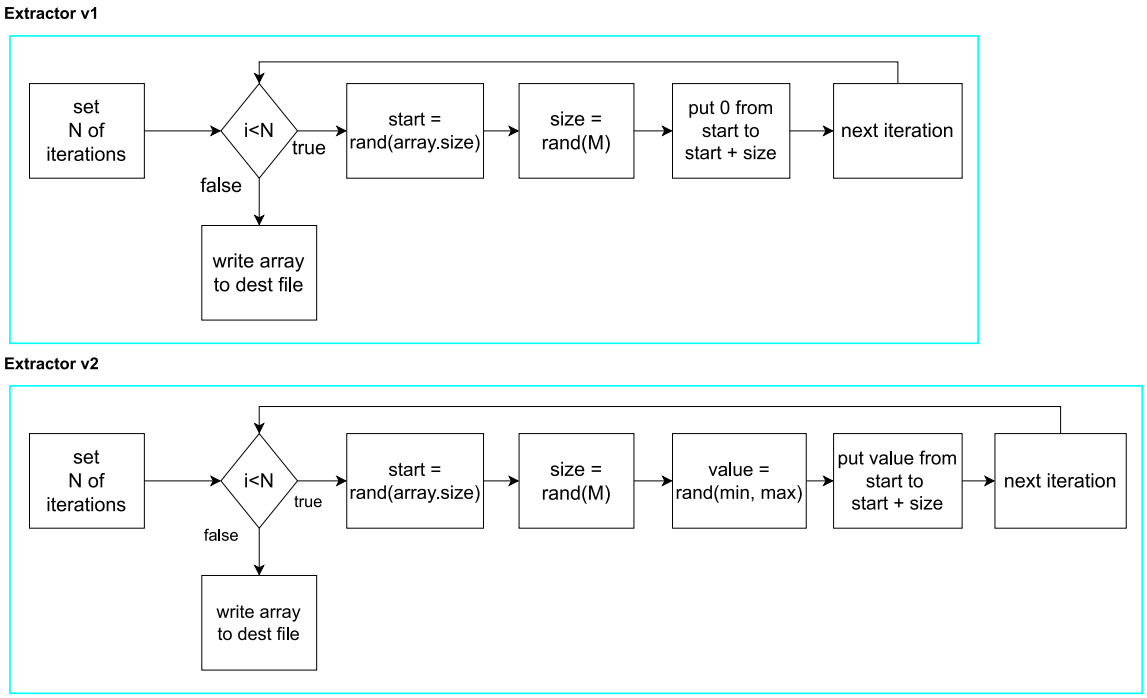
N je počet opakovaní algoritmu nad zvukovým signálom

start je začiatok úseku (index vzorky)

size je dĺžka úseku (vo vzorkách)

value je hodnota, ktorou sa vyplní prázdny úsek (spodný panel na obrázku 4.15)

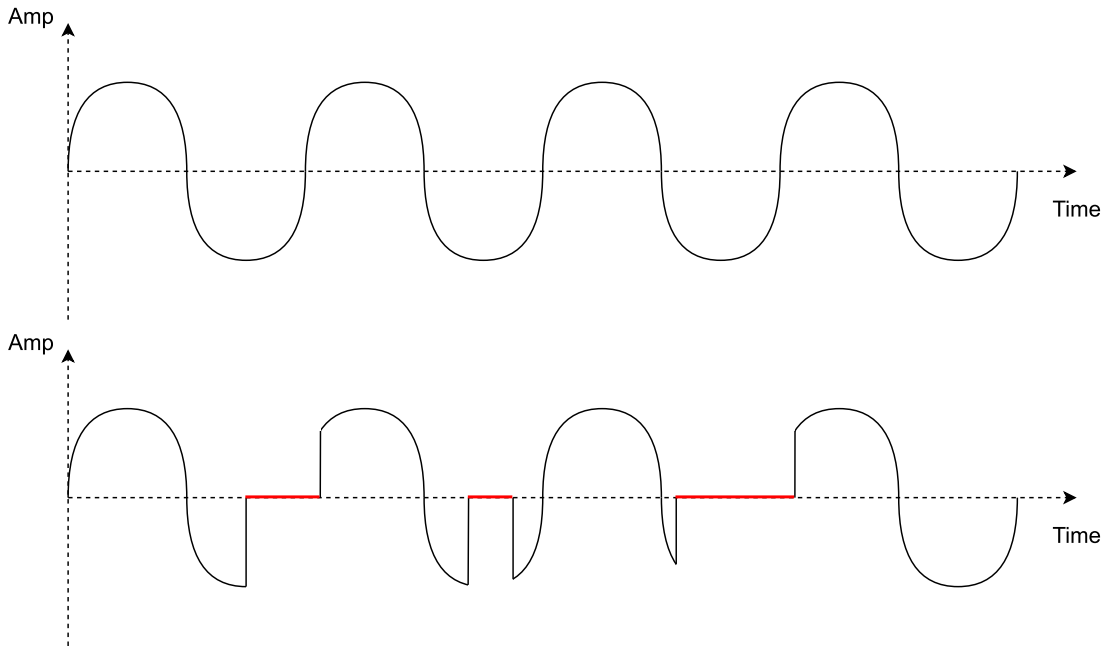
Z týchto parametrov sú 2 nastaviteľné, a to *N* (v programe ako Intensity) a *size* (Width). Ostatné parametre sú určené náhodne pomocou generátora náhodných čísiel.



Obrázek 4.15: Schéma extractor efektu.

Pri nastavovaní parametrov je miera efektu priamo úmerná hodnotám. Zvýšením počtu opakovaní sa dá dosiahnuť odstránenie vzoriek z viacerých pozícií a zvýšením parametru *size* sa dajú zväčšiť úseky odstraňovaných vzoriek.

Jednou z vlastností efektu je však to, že pracuje s určitou mierou náhodnosti. Napríklad začiatočný index algoritmu je vždy volený úplne náhodne, veľkosť úseku je volená náhodne od 0 do *size* a v druhej verzii algoritmu je hodnota vyplňujúca úsek volená náhodne z intervalu od minimálnej po maximálnu hodnotu daného signálu.



Obrázek 4.16: Zvuková vlna pred / po úprave cez Extractor.

4.7.2 Reverz

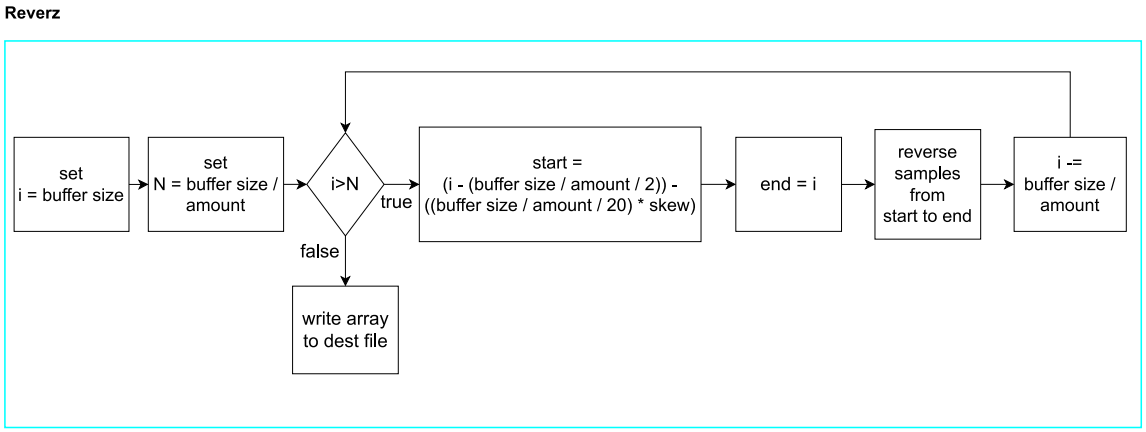
Reverz je efekt spracovávajúci signál v časovej doméne. Obsahuje algorimtus pre zmenu usporiadania určitých vzoriek signálu a to reverzovaním určitých častí signálu.

Efekt obsahuje 2 parametre:

Amount je veľkosť úsekov spracovaných prvou časťou algoritmu

Skew je veľkosť reverzovanej časti signálu v rámci jedného úseku

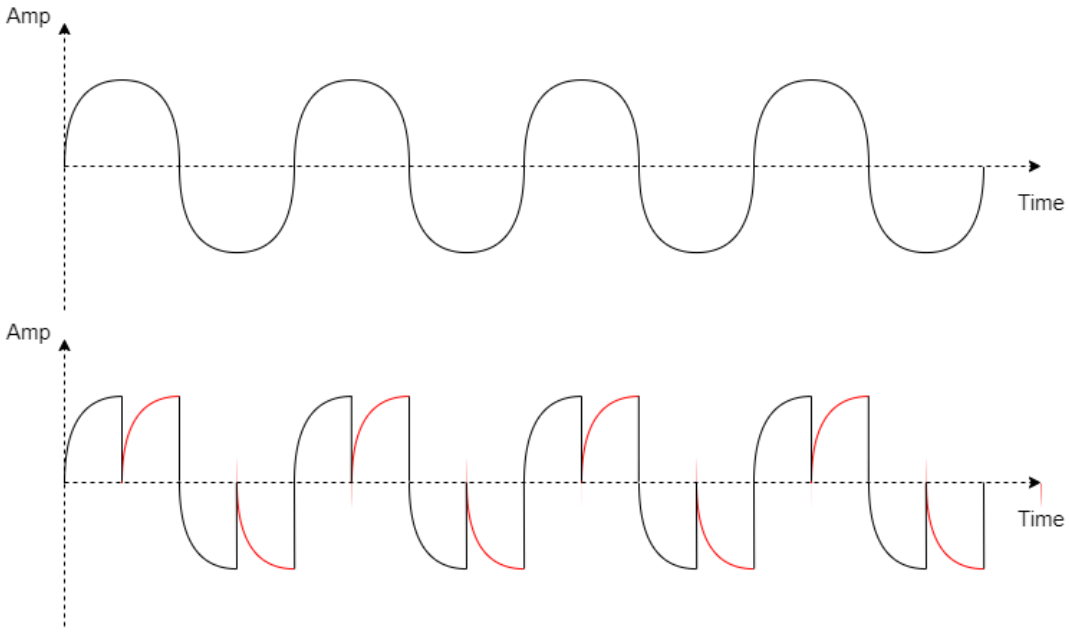
Reverz pracuje na základe algoritmu na obrázku 4.17.



Obrázek 4.17: Schéma Reverz efektu.

Algoritmus implementuje preusporiadanie spracovaného signálu. Signál je rozdeľovaný na niekoľko častí podľa parametru `Skew`, ktoré obsahujú rovnaký počet vzoriek. Tieto časti

sú postupne prechádzané a vzorky v týchto úsekov sú reverzne preusporiadané. Pri prechádzaní signálu však nie sú reverzované všetky úseky, ale iba každý druhý, ako je uvedené aj na obrázku 4.18.



Obrázek 4.18: Zvuková vlna pred / po úprave cez Reverz.

Reverzované úseky majú v základe rovnakú veľkosť, ako úseky, ktoré nie sú spracované. Túto veľkosť je možné ovplyvniť zmenou parametra *Skew*. Jeho pôvodná hodnota je 0, ale je možné ju meniť v intervale $(-10, 10)$ a to s krokmi o veľkosti 1. Záporné hodnoty vedú k zmenšeniu reverzovaných úsekov a kladné k ich zväčšeniu. Zmena veľkosti však funguje ako zmena pomeru a to znamená, že zmenou reverzovaných úsekov sa nepriamo úmerne mení aj dĺžka nespracovaných častí.

4.7.3 Stutter

Stutter (tzv. "koktanie") je efekt, ktorý je zložený z dvoch spolupracujúcich častí. Efekt vznikol kombináciou vlastného efektu, ktorý pracuje v časovej doméne signálu a existujúceho efektu Chorus. Komponenty efektu Stutter sú popísané na obrázku 4.19.

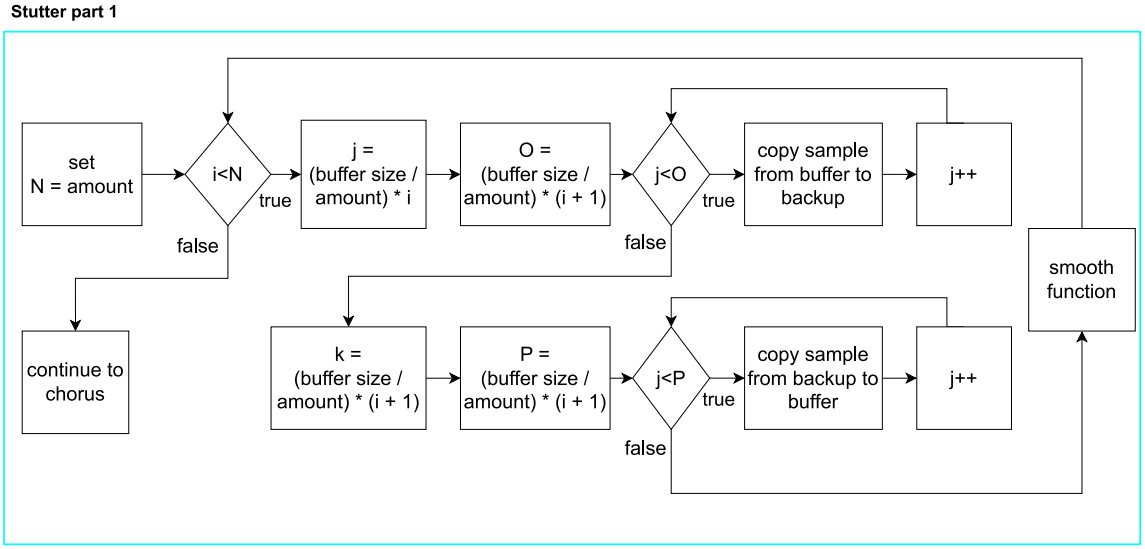
Efekt obsahuje 2 parametre:

Amount je veľkosť úsekov spracovaných prvou časťou algoritmu

Chorus je úroveň Chorus efektu

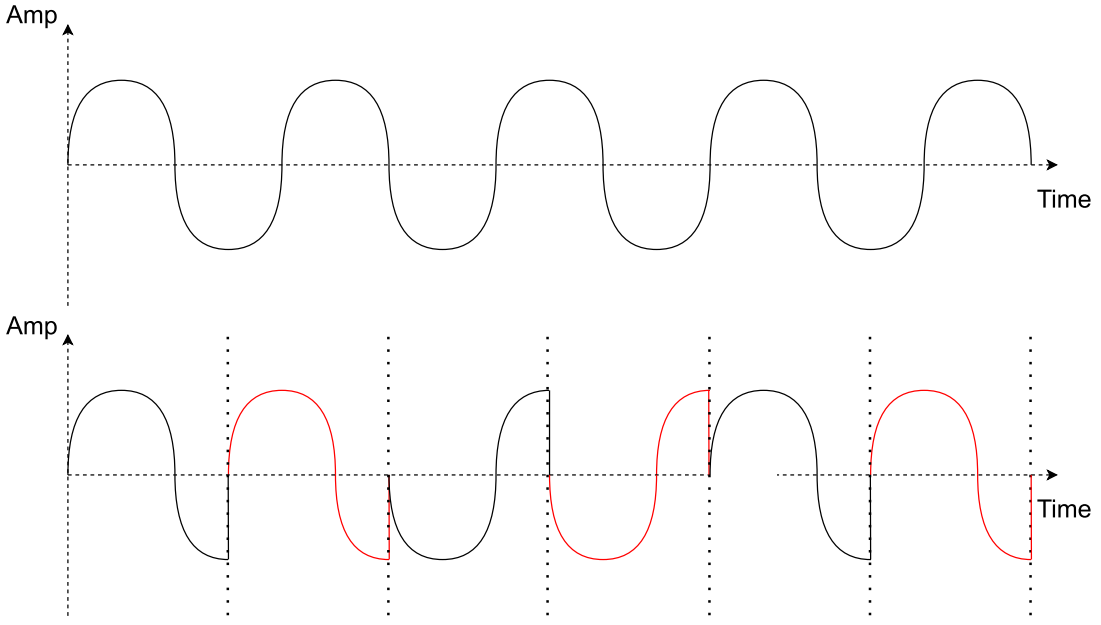
Prvá časť efektu rozdeľuje signál na určitý počet častí, ktorý je rovnaký, ako parameter *Amount*. Vzhľadom na rovnomerné rozdelenie signálu má každá časť rovnakú dĺžku a tieto časti sú vždy spracovávané v pároch, od čoho sa závisí aj to, že parameter *Amount* môže nadobúdať len párne hodnoty. Pri prechádzaní signálu algoritmom je prvá časť v každom páre vždy kopírovaná a následne pri spracovaní druhej časti z páru je obsah druhej časti nahradený obsahom prvej časti. Prvá časť efektu je znázornená na obrázku 4.20.

Vo výsledku prvá časť efektu vytvára u poslucháča efekt "zasekávania" zvukovej stopy, ktorý môže pripomínať prehrávanie z chybného zariadenia alebo z chybného zvukového



Obrázek 4.19: Schéma Stutter efektu.

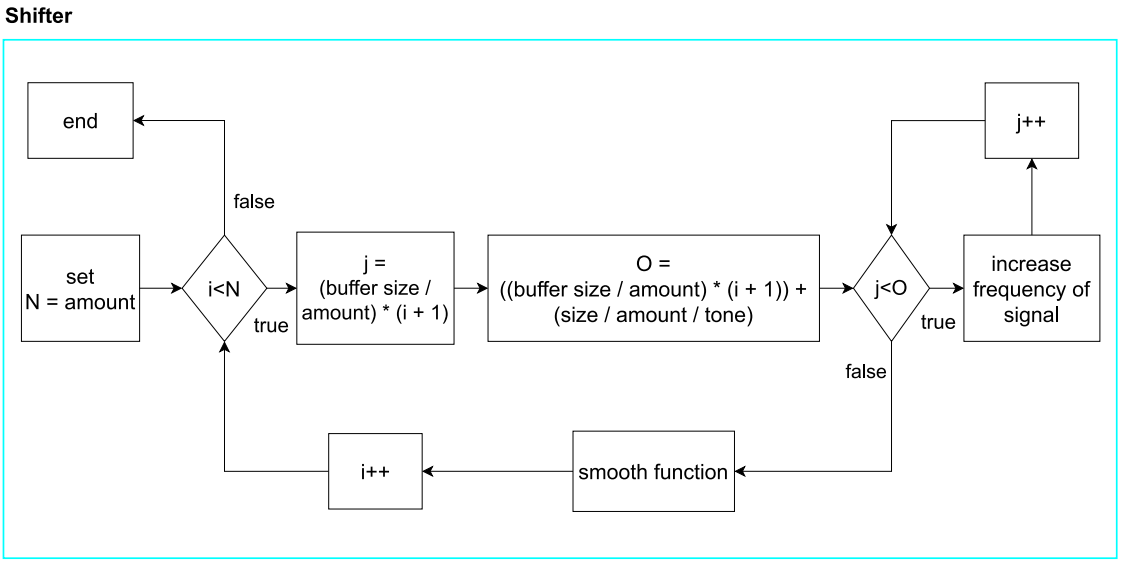
média. Tento efekt je možné využiť pre zvýšenie alebo zmenu rytmickosti zvukovej stopy. To znamená, že vo zvukoch, ktoré majú určitý výrazný rytmus môže byť tento rytmus zmenený, alebo vo zvukoch, ktoré majú len jemný a nevýrazný rytmus môže byť tneto rytmus zvýraznený.



Obrázek 4.20: Zvuková vlna pred / po úprave cez Stutter.

4.7.4 Shifter

Shifter je efekt kombinujúci úpravu signálu v časovej a frekvenčnej doméne pozostávajúci z dvoch častí. Prvá časť pracuje na rovnakom princípe, ako pri efekte Stutter, a to rozdeľovaním signálu na pravidelné úseky a druhá časť upravuje frekvenciu signálu v častiach týchto úsekov.



Obrázek 4.21: Schéma Shifter efektu.

Efekt obsahuje 2 parametre:

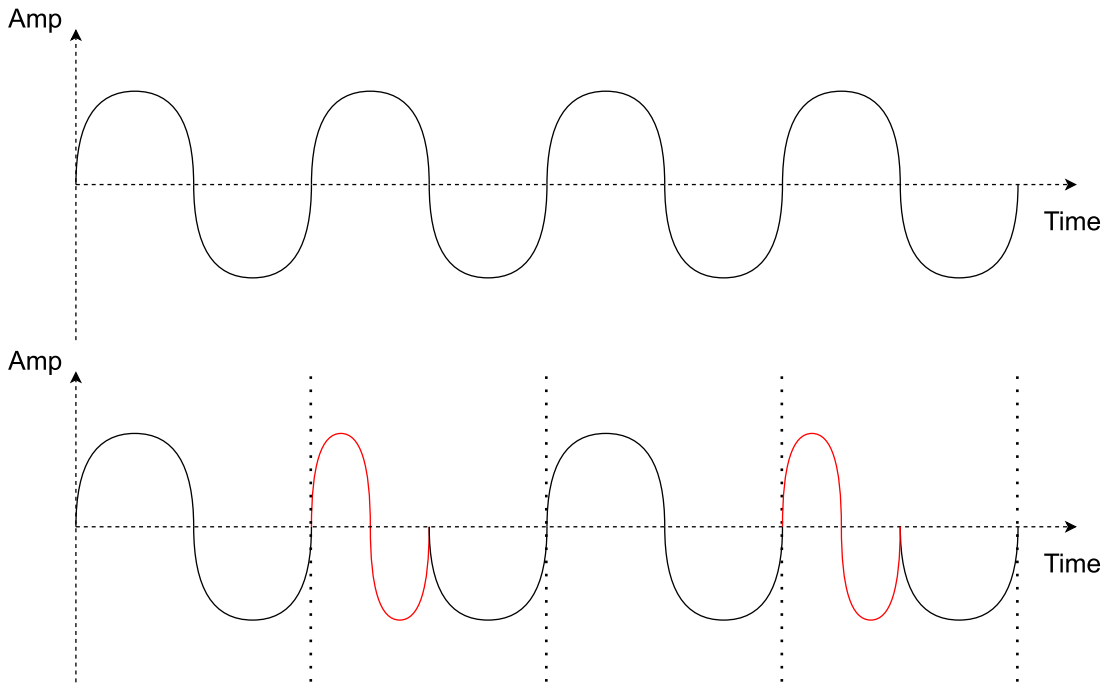
Amount je veľkosť úsekov spracovaných prvou časťou algoritmu

Tone je koeficient, ktorým je zvyšovaná frekvencia vybraných úsekov

Pri rozdeľovaní na jednotlivé úseky je frekvencia zvyšovaná od začiatku druhej polovice v každom úseku. Vzhľadom na to, že zvyšovaním frekvencie sa kráti dĺžka signálu, sa veľkosť tejto časti úseku, v ktorom je zvyšovaná frekvencia skracuje v závislosti na parametri *Tone*. Napr. pri hodnote 2.0 je frekvencia zvýšená 2-krát a tým pádom aj upravená časť má 2-krát nižšiu veľkosť (vidno na obrázku 4.22).

Algoritmus pre zvyšovanie frekvencie je identický s algoritmom v efekte Pitch 4.6. Jediný rozdiel je v obmedzení len na zvýšenie frekvencie, z toho dôvodu je parameter *Tone* v rozsahu (1.0, 8.0).

Tento efekt je možné využiť pre jemné výškové variácie v zvukových stopách s bicími nástrojmi a podobnými zvukmi z tejto kategórie, čomu pomáha aj pravidelné rozdelenie spracovaných úsekov. Ďalším využitím sú melodické a vokálové slučky a to takisto z dôvodu pridania variácii. Efekt je však využiteľný aj pre vytváranie glitch efektu a to pri nastavení hodnôt parametrov *Amount* a *Tone* na pomerne vysoké hodnoty.



Obrázek 4.22: Zvuková vlna pred / po úprave cez Shifter.

Kapitola 5

Implementačné prostriedky

5.1 Framework JUCE

Framework pre jazyk C/C++ určený pre tvorbu hudobných efektov a programov pre úpravu a spracovanie zvukových signálov. Je dostupný v 4 edíciách: JUCE Personal, JUCE Indie, JUCE Pro, JUCE Education.

Edície sa líšia podľa účelu použitia a podľa osoby/ firmy, ktorá tento framework využíva, pričom sa berie do úvahy výnos danej osoby/ firmy za jeden fiškálny rok. Pri tvorbe nášho programu bola využitá licencia typu „Personal“, ktorá obmedzuje výnos z prípadného predaja softvéru na 50.000$ za jeden rok. Daná licencia je zadarmo a pre naše účely je dostačujúca. Jediná nevýhoda spočíva v nutnosti zobrazovať vyskakovacie okno s textom „Made with JUCE“ pri každom spustení programu [16].

JUCE sa používa vo forme aplikácie Projucer, ktorá slúži na vytvorenie a spravovanie projektu. Pri vytváraní projektu si užívateľ zvolí cieľový operačný systém, na ktorom chce vyvíjať daný projekt a parametre projektu. Projekt sa vytvorí pre určité vývojové prostredie v závislosti od zvoleného OS. Napríklad pre Windows je to Visual Studio. Okrem nastavenia samotného projektu si zvolí knižnice, ktoré chce v projekte využívať. Knižnice sa volia manuálne, alebo je možnosť si zvoliť typ aplikácie a podľa toho sa vyberú určité knižnice potrebné pre daný typ projektu.

JUCE ponúka knižnice pre prácu so zvukovými signálmi, zvukovými vstupnými a výstupnými zariadeniami a tiež pre tvorbu grafického užívateľského rozhrania a videí.

JUCE je využiteľný hlavne pre tvorbu hudobných efektov v plugin formáte, ako VST, VST3, AU, AUv3, AAX a LV2, ale aj pre samostatné multi-platformové hudobné programy pre Windows, Mac a Linux. Výhodou JUCE je aj zabezpečenie základných súčastí aplikácie potrebných pre komunikáciu aplikácie so systémom a vstupnými / výstupnými zariadeniami. Pri úprave zvukového signálu je možné pristupovať priamo k jednotlivým vzorkám signálu v bufferi, ktoré je možné upravovať cez vlastné metódy, alebo cez preddefinované metódy z knižníc framework-u [14].

5.2 Visual Studio

Visual Studio je vývojové prostredie určené pre vývoj grafických aj textových aplikácií. Pre vývoj nášho programu bola použitá verzia `Visual Studio Community 2019`.

Visual Studio obsahuje všetky potrebné nástroje pre tvorbu zdrojového kódu, vrátane formátovaného editora a funkcie `Intellisense` pre nápovede a dopĺňanie príkazov. Ďal-

šími použitými nástrojmi sú aj profilovače výkonu aplikácie a užívateľskej pamäte. Visual Studio poskytuje aj rozdielne vytváranie výslednej aplikácie pre účely testovania (rýchlejšia kompilácia) a konečného vydania (možnosť optimalizácií).

5.3 Inno Setup

Inštalačný súbor výsledného programu odoslaný na užívateľské testovanie bol vytvorený pomocou programu Inno Setup. Je to voľne dostupný program určený pre vytváranie inštalačných balíkov pre aplikácie. Program pracuje na báze užívateľom definovaných skriptov, ktoré obsahujú inštrukcie a cesty k súborom pre vytvorenie inštalačného balíka.

Inštalačný súbor nášho programu obsahuje výsledný program v `exe` formáte a priečinok s presetmi. Definovaný skript pre inštaláciu je možné otvoriť priamo s Inno Setup programom a ihneď spustiť. Vytvárať inštaláciu je tak možné okamžite po každej kompilácií.

5.4 Hardvér

Pri vývoji a testovaní nášho programu bol použitý laptop Lenovo Legion Y530 s procesorom Intel Core i7-8750H, užívateľskou pamäťou 8GB a operačným systémom Windows 10 64-bit. Ďalším hardvérom boli náhlavné monitorovacie slúchadlá Sony MDR-7510 s uzavretou konštrukciou. Okrem slúchadiel nebol použitý iný hardvér prispôsobený špeciálne pre monitorovanie zvuku, ako napr. monitorovacie reproduktory alebo externá zvuková karta.

Kapitola 6

Implementácia programu

6.1 Základy implementácie

Funkcionalita je implementovaná v triedach, a to tak, že každý efekt má svoju vlastnú triedu. Vedlajšie prvky, grafické rozhranie a prepojenie grafického rozhrania s efektami sú implementované zvlášť.

Pri implementácii jednotlivých efektov je v každej triede dodržaný určitý postup pre uľahčenie a sprehľadnenie písaného zdrojového kódu a ľahkú úpravu algoritmov v prípade rozširovania programu.

Každý efekt má svoje vlastné ovládacie prvky, ktoré slúžia na nastavenie parametrov efektov.

6.1.1 Buffer

Dôležitým prvkom je tzv. buffer. Buffer je dátová štruktúra, ktorá udržuje jednotlivé vzorky signálu vo forme jednoduchého 2D poľa (1 pole pre každý kanál) a zároveň aj parametre signálu. Najdôležitejšie parametre sú veľkosť bufferu (vo vzorkách), počet kanálov a veľkosť vzoriek.

	#0	#1	#2	#3	#4	#5	#6	#7	#8
CH 0	0	1	0.5	-0.002	-0.3	0.21	0.56	0.15	-0.145
CH 1	0	0.44	0.65	.0.22	-0.247	-0.781	-0.37	0	0.13

Number of samples	Number of Channels	Pointer
9	2	*

Obrázek 6.1: Schéma štruktúry buffer

Jednotlivé hodnoty sú hodnoty vzoriek. Hodnoty vzoriek majú rozsah podľa použitého dátového typu. Štandardne je používaný 32 bitový float, ale používané sú aj dva ďalšie typy, a to 24 bitový a 16 bitový int. Každá vzorka sa nachádza v kanáli s určitým indexom a má svoj vlastný poradový index v rámci kanála. Buffer zároveň udržiava ukazovateľ na prvú

vzorku v kanáli, pomocou ktorého je možné iterovať cez všetky vzorky, ale cez jeho metódy je možné získať naraz celé pole ukazovateľov na vzorky (v efektoch je väčšinou využívaná práve táto metóda).

6.2 Načítanie a uloženie zvukovej stopy

Keďže program pracuje v offline režime, tak zvuková stopa sa načítava zo súboru a výsledok po úprave sa musí uložiť do súboru.

Načítavanie je jednoduché, riešené cez okno Windows prieskumníka. Možné formáty súborov nie sú nijak limitované, pretože ich množstvo je vysoké a čítač súborov z JUCE dokáže načítať každý formát obsahujúci zvukový signál. To znamená, že užívateľ sa môže pokúsiť otvoriť aj nezvukové súbory, čo skončí neúspechom, ale predpokladáme určitú inteligenciu užívateľa.

Pri ukladaní súboru je súbor na uloženie vybraný z výstupného buffer-u (teda z buffer-u, ktorého obsah je zároveň vykresľovaný v okne pre zobrazenie zvukovej vlny) a následne je spojený s atribútmi o danom súbore. Užívateľ si môže po otvorení okna Windows prieskumníka zvoliť názov a formát a zvuková stopa aj s atribútmi sa pri uložení prevedie do daného formátu. Prieskumník sa štandardne otvorí v adresári, odkiaľ bola načítaná zdrojová stopa, ale názov súboru je potrebné zadať ručne.

Načítanie aj uloženie súboru sú implementované s pomocou tutoriálu z JUCE frameworku [17].

6.3 Vizualizácia amplitúdy zvuku

V moderných hudobných programoch je dôležité, aby užívateľ dostával vizuálnu odpoveď od programu, kvôli tomu, aby mal lepšiu a presnejšiu predstavu o upravovanej zvukovej stope.

V programe je implementované zobrazenie zvukovej vlny spracovaného súboru. Zvuková vlna je zobrazená celá v jednom okne. Toto okno zobrazuje v podstate graf hodnoty (y os) v čase (x os). V závislosti od počtu kanálov je zobrazená 1 (mono) alebo 2 (stereo) zvukové vlny. Zvuk s vyšším počtom kanálov, než 2 nie je možné plnohodnotne zobraziť.

Vizualizér amplitúdy zvukovej vlny je implementovaný s pomocou tutoriálu z JUCE frameworku [18].

Obrázek 6.2: Zobrazenie zvukovej vlny.

6.3.1 Priblíženie zobrazenia

V lepších vizualizéroch zvukových stôp zvykne byť k dispozícii aj priblíženie zobrazenia spracovávanej zvukovej vlny. Priblíženie je využiteľné hlavne, ak užívateľ potrebuje presnejšie vidieť niektoré miesta vo zvukovej stope, hlavne pri práci s efektami, ktoré pracujú so

zvukovým signálom v časovej doméne (keďže zvuková vlna je zobrazovaná v časovej doméne). V programe je možné približovať signál len z pohľadu času, t.j. približovanie na osi amplitúdy nie je implementované.

Pri zobrazení je používaná funkcia `drawChannels()`, ktorá vykresľuje časť načítanej zvukovej stopy v určenom intervale. Pri základnom nastavení má interval rozsah (0, dĺžka signálu), čiže stopa je zobrazená celá.

Pri približovaní a oddialení zobrazenia je upravovaný práve daný interval a to nasledovne:

$$(0 + K_z, L_a - K_z) \tag{6.1}$$

kde:

L_a je dĺžka zvukovej stopy v sekundách

K_z je koeficient, ktorý mení dolnú a hornú hranicu intervalu

Približovanie na danom princípe však nezohľadňuje konkrétne miesto v signále, ktoré má byť priblížené, pretože koeficient K_z je rovnaký pre obidve hranice intervalu. Preto je použitý výpočet doplnený aj o koeficient K_p, ktorý posúva hranice intervalu.

$$K_p = \begin{cases} K_p + K_z \cdot Z_p & \text{pre } Z_p < \frac{1}{2} \\ K_p + K_z \cdot (2Z_p - 1) & \text{pre } Z_p > \frac{1}{2} \end{cases} \tag{6.2}$$

kde:

K_p je koeficient pre posun hraníc vyššie / nižšie

Z_p je pozícia kurzora myši v okne vizualizéra (napr. hodnota 0.5 znamená pozíciu v strede okna)

Koeficient K_p sa po získaní pripočíta k intervalu nasledovne:

$$(0 + K_z + K_p, L_a - K_z + K_p) \tag{6.3}$$

Podobný výpočet je implementovaný aj pre oddialenie priblíženého zobrazenia.

Nevýhodou približovania je však aj limitácia zo strany JUCE frameworku, pretože rozlíšenie zobrazenej zvukovej stopy je pomerne nízke a preto je výsledná kvalita pri väčších hodnotách priblíženie tiež nízka. Pre účely programu je však dostatočná, pretože podrobná inšpekcia zvukovej vlny signálu nie je cieľom tejto aplikácie.

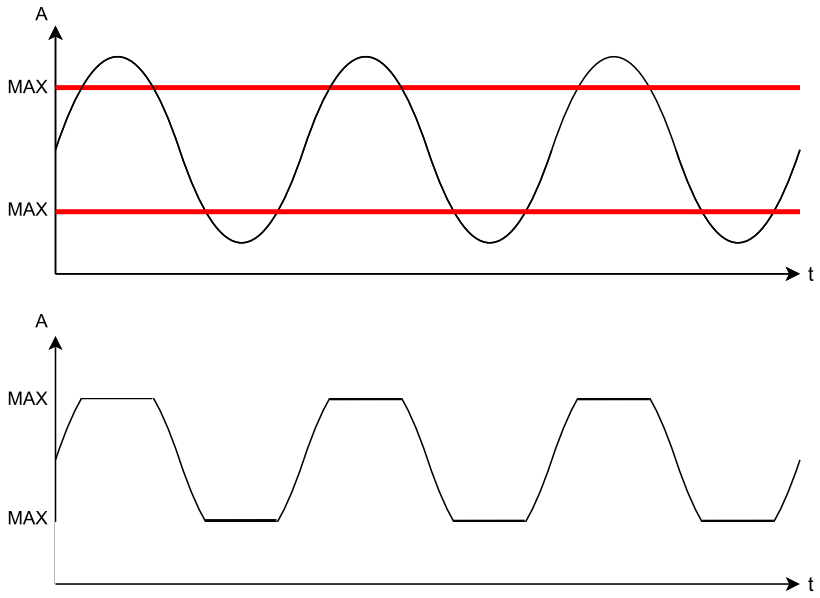
6.3.2 Clipping

Ďalším vizualizačným prvkom je indikácia tzv. clipping-u. Clipping nastáva v prípade, ak hociktorá vzorka, alebo úseky vzoriek presiahnu maximálnu možnú hodnotu. V takomto prípade nedokáže audio výstup presne prehrať takúto vzorku, keďže na jej výrobu je potrebný vyšší výkon, než audio výstup dokáže vyprodukovať [5].

Preto je dôležité vedieť, či pri úprave zvukovej stopy nedochádza k tomuto efektu. Detegovať clipping sa dá aj graficky vyčítaním z vizualizovanej zvukovej vlny, ale tento spôsob nie je vždy presný a spoľahlivý.

Zistený clipping je v tomto programe znázornený jednoduchým farebným prvkom (textový štítok), ktorý je zelený s textom “No Clipping” ak clipping nie je prítomný a červený s textom “Clipping!!!”, ak je clipping detekovaný.

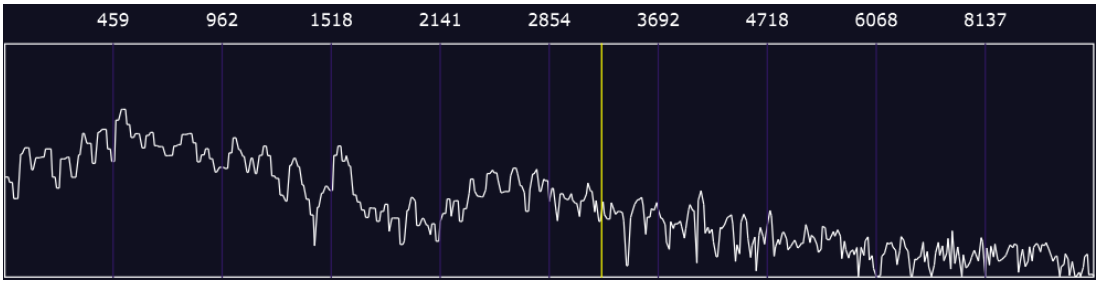
Clipping má na zvukový signál podobný efekt, ako hard-clipping distortion efekt (4.4.1) :



Obrázek 6.3: Clipping zvukovej vlny.

6.4 Vizualizácia frekvenčného spektra

Doplnkom k vizualizácii amplitúdy zvukového signálu je zobrazenie frekvenčného spektra prehrávaného zvuku. Zobrazenie frekvenčného spektra je užitočné hlavne pre lepší prehľad o zastúpení jednotlivých frekvenčných zložiek v signále a o ich amplitúde. Využiteľné je zároveň aj pri práci s efektami, ktoré sú založené na frekvenčnej modulácii signálu (napr. zmena výšky zvuku, chorus, flanger...), alebo s efektami, ktoré upravujú frekvenčné spektrum (napr. filter, ekvalizér, atď).



Obrázek 6.4: Zobrazenie frekvenčného spektra.

Vizualizácia je realizovaná cez funkcie, ktoré využívajú FFT algoritmus pre rozloženie signálu na jeho jednotlivé frekvenčné zložky. Jednotlivé algoritmy sú implementované na základe JUCE tutoriálu [15].

Algoritmus funguje na princípe postupného ukladania aktuálne prehrávaných vzoriek, následného výpočtu jednotlivých frekvencií a ich vykresľovaní do okna pre vizualizáciu frekvenčného spektra.

Prvá časť načítava vzorky po blokoch, pretože aj pri prehrávaní sa signál načítava po blokoch. Tieto vzorky sú postupne spracovávané po blokoch, avšak pri spracovaní je už veľkosť blokov určená veľkosťou okna použitej okienkovej funkcie.

Ďalšia časť vykonáva transformáciu pomocou FFT s použitím tzv. Hann-ovej okienkovej funkcie, pri ktorej sú vypočítané hodnoty amplitúdy jednotlivých frekvenčných zložiek a tieto hodnoty sú postupne vykreslené na príslušné miesta podľa použitého rozloženia. Veľkosť okna je pevne nastavená na hodnotu 4096 vzoriek. Frekvenčné spektrum má obmedzené rozlíšenie určené obmedzenou veľkosťou štruktúry `scopeData[]`. Veľkosť danej štruktúry je nastavená na hodnotu 512 vzoriek. Zväčšenie tejto štruktúry znamená vyššie rozlíšenie, ale zároveň aj vyššiu výpočtovú záťaž.

6.4.1 Rozloženie frekvenčného spektra

Poloha vykresľovaných vzoriek sa vždy riadi určitým rozložením, ktoré môže byť napr. lineárne alebo logaritmické. Lineárne rozloženie je jednoduché na implementáciu, ale má nevyhodu kvôli tomu, ako ľudský sluch vníma jednotlivé frekvenčné pásma. Sluch človeka je lepšie prispôsobený pre rozlišovanie nižších a stredných frekvencií, pretože práve tam sa nachádza najväčšie množstvo zvukov vyskytujúcich sa v prírode. To znamená, že je vhodné vybrať rozloženie, ktoré má vyššie rozlíšenie v nižšej časti spektra, ktoré sa postupne znižuje pri prechode do vyšších frekvencií, čo spĺňa práve logaritmické rozloženie. Rozdiel medzi lineárnym a logaritmickým rozložením je uvedený na obrázkoch 6.5 a 6.6.

Rozloženie použité v programe sa riadi nasledovnou rovnicou:

$$X_{log} = -K \ln(1 - X_f) \tag{6.4}$$

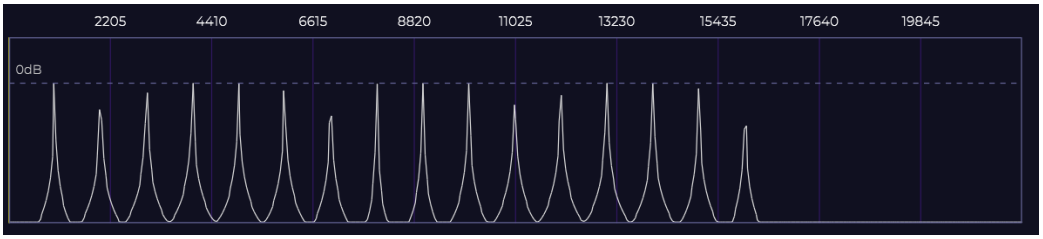
kde:

X_{log} je X súradnica s upraveným logaritmickým rozložením

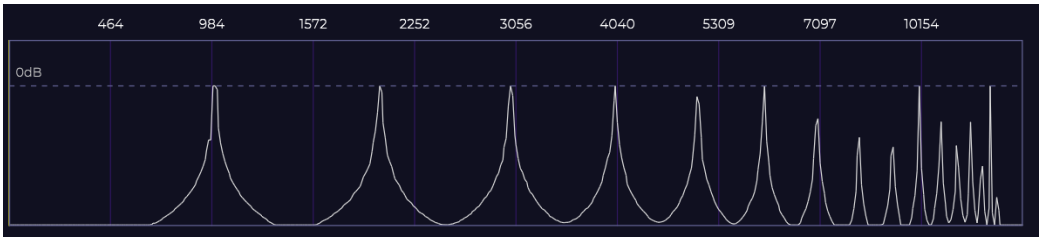
X_f je pôvodná X súradnica frekvenčnej zložky

K je koeficient, ktorý mení rozloženie

Výhodou použitého rozloženia je možnosť zmeny rozloženia pomocou zmeny koeficientu K. Pri nižšom koeficiente zaberá viac miesta nízkofrekvenčné pásmo a pri vyššom koeficiente je možné priblížiť vysokofrekvenčné pásmo.



Obrázek 6.5: Lineárne rozloženie frekvenčného spektra.



Obrázek 6.6: Logaritmické rozloženie frekvenčného spektra.

6.4.2 Indikácia vybranej frekvencie

Vizualizér obsahuje aj viacero značiek, ktoré zobrazujú frekvencie na určitých miestach v spektre (vidno na obrázku 6.4). Značky sú pevne umiestnené v pravidelných úsekoch a vzhľadom na to, že rozloženie frekvencií je možne meniť, je nutné meniť aj ich hodnoty, ktoré sú aktualizované pri každej zmene rozloženia. Ich hodnoty sa vypočítavajú pomocou funkcie `cordValueToFreq()`, ktorá konvertuje X súradnicu značky na hodnotu frekvencie pomocou rovnice 6.4 doplnenej o vynásobenie 1/2 vzorkovacej frekvencie.

Ďalším pomocným prvkom je indikátor konkrétnej frekvencie. Indikátor je možné umiestniť podľa potreby na určité miesto v rámci okna a jeho hodnota je následne pri každej zmene umiestnenia zobrazená vedľa okna vizualizéra. Hodnota frekvencie v mieste indikátora je takisto prispôsobená zmenám rozloženia a je vypočítavaná rovnakým spôsobom, ako frekvenčné značky.

Indikátor slúži pre presné sledovanie určitej frekvencie v určitom bode spektra, čo je využiteľné pri sledovaní konkrétnych kritických frekvencií pri úprave zvukovej stopy.

6.4.3 Frekvenčná odozva filtra

Ďalším prvkom frekvenčného okna je zobrazenie frekvenčnej odozvy všetkých filtrov, ktoré sú v programe implementované. Frekvenčná odozva je zobrazovaná zvlášť pre každý filter a krivky odozvy sú farebne odlíšené. Krivky sú vykresľované, len ak je daný filter aktívny, a to kvôli zníženiu výpočtovej záťaže programu.

magnitudeResponse<filter>()

Pre zobrazenie krivky odozvy je potrebné vedieť magnitúdu frekvenčnej odozvy pre každú frekvenciu frekvenčného spektra. Magnitúda frekvencie je v tomto prípade koeficient, ktorým filter upravuje danú frekvenciu (napr. magnitúda s hodnotou 1.0 znamená, že filter neupravuje hodnotu danej frekvencie).

Práve magnitúda frekvenčnej odozvy pre každú frekvenciu je vypočítavaná pomocou funkcie `magnitudeResponse()`. Funkcia využíva základný vzorec pre výpočeť magnitúdy odozvy pre Biquad filter (iné typy majú iný spôsob výpočtu), ktorý vyzerá nasledovne:

$$|H(e^{j\omega})| = \sqrt{\frac{a_0^2 + a_1^2 + a_2^2 + 2(a_0a_1 + a_1a_2)\cos\omega + 2a_0a_2\cos 2\omega}{1 + b_1^2 + b_2^2 + 2(b_1 + b_1b_2)\cos\omega + 2b_2\cos 2\omega}} \tag{6.5}$$

kde:

$|H(e^{j\omega})|$ je výsledný koeficient magnitúdy

a_x, b_x sú koeficienty daného filtra

ω je normalizovaná frekvencia v radiánoch

Normalizovanú frekvenciu ω je nutné získať pred výpočtom samotnej magnitúdy a to nasledovne:

$$\omega = 2\pi\frac{f_c}{F_s} \tag{6.6}$$

kde:

f_c je frekvencia v Hz, pre ktorú je magnitúda vypočítaná

F_s je použitá vzorkovacia frekvencia v Hz

Funkcia je použitá v metóde `drawFilterResponse()`, ktorá vykresľuje krivku medzi každou 8. frekvenciou od $f_c = 0$ do $f_c = F_s/2$, pretože vykresľovať všetky body krivky predstavuje vysokú záťaž na výkon programu, ak sú filtre aktívne a pre dôveryhodné zobrazenie to nie je potrebné. Podobne aj magnitúda krivky nie je zobrazená presne, ale s tým, že hodnota 1.0 sa nachádza v polovici Y rozmeru okna. Pre účely zobrazenia je dôležitý hlavne tvar frekvenčnej krivky, ktorý je vykresľovaný s dostatočnou presnosťou.

6.5 Prehrávanie zvuku

Prehrávanie zvuku je ovládané dvomi ovládacími prvkami - tlačidlo Play a Stop. Play slúži na spustenie prehrávania a Stop na úplné zastavenie. Tlačidlo Play má zároveň aj funkciu Pause, ktorá je implementovaná kvôli možnosti pozastaviť prehrávanú stopu.

Funkcia Pause je aktívna iba počas prebiehajúceho prehrávania zvuku a prehrávaný zvuk je po prerušení prehrávania touto funkciou ponechaný na aktuálnej dosiahnutej pozícii. Funkcia Play je aktívna iba počas pozastaveného prehrávania a to znamená, že v programe je možné využiť 1 grafický prvok pre realizáciu obidvoch funkcií.

Funkcia Stop narozdiel od Pause vráti prehrávanie na začiatok stopy.

Prehrávanie súboru je implementované s pomocou tutoriálu z JUCE frameworku [17].

6.5.1 Indikátor pozície prehrávania

Indikátor pozície (na obrázku 6.2) dáva užívateľovi vizuálnu odozvu ohľadom aktuálnej pozície, na ktorej sa prehrávanie zvukovej stopy nachádza. Indikátor je implementovaný ako jednoduchá zvislá čiara, ktorej pozícia je vypočítavaná na základe aktuálne prehrávanej vzorky a celkovej dĺžky zvukovej stopy a to nasledovne:

Najprv je vypočítaná aktuálna pozícia prehrávania v sekundách z pozície vzorky a vzorkovacej frekvencie:

$$p_s = \frac{p_x}{F_s}$$

kde:

p_x je pozícia vzorky (vo vzorkách)

F_s je vzorkovacia frekvencia v Hz

Následne je vypočítaná súradnica X_C v okne, kde bude indikátor vykreslený:

$$X_C = \frac{p_s}{L_a} \cdot W_w + W_X$$

kde:

L_a je dĺžka audia v sekundách

W_w je šírka okna pre vykreslenie

W_X je súradnica X ľavého okraja okna

Výpočty polohy sú vykonávané rýchlosťou danou programovým časovačom 6.5.3. Polohu indikátora je zároveň možné ovládať aj pomocou funkcií pre posun prehrávania 6.5.2.

6.5.2 Posun pozície prehrávania

Pre ovládanie prehrávania sú k dispozícii aj ďalšie dve funkcie - Forward a Backward, ktoré slúžia na posun pozície prehrávania. Pozícia je posúvaná na základe použitej funkcie (Forward - ďalej, Backward - späť v čase). Veľkosť posunu sa počíta vo vzorkách a je možné ju nastaviť na vlastnú hodnotu. Táto hodnota je následne pripočítaná / odčítaná od parametru pozície, ktorý je využívaný pri prehrávaní zvukovej stopy vo funkcii `getNextAudioBlock` 6.5.4 a tiež pri výpočte polohy indikátora pozície.

Vzhľadom na to, že indikátor sa vykresľuje v závislosti od parametra, ktorý určuje pozíciu pre prehrávania pre prehrávač zvuku, je možné meniť polohu aj pomocou kliknutia na určitú časť okna. Pri kliknutí sú spracované súradnice kliknutia a tieto sú použité spolu s veľkosťou vykresľovacieho okna a dĺžkou načítaného zvuku pre výpočet novej pozície prehrávania. Indikátor je následne aktualizovaný a presunutý na danú pozíciu. Posun indikátora pomocou kliknutia je inmplementovaný hlavne pre pohodlné presúvanie prehrávaného zvuku, pretože je to praktickejšie, než posun pomocou funkcii Forward a Backward. Tieto dve funckie slúžia skôr na kritický presný posun po určitom množstve vzoriek.

6.5.3 Timer

V programe je pre účely prehrávania a správneho zobrazovania grafických prvkov počas prehrávania implementovaný časovač.

Časovač je inicializovaný pri spustení programu, ale aktívny je až počas prehrávania zvuku. Prehrávač zvuku má k dispozícii stavovú premennú definujúcu, v ktorom stave prehrávania sa aktuálne nachádza. Stavová premenná môže nadobúdať 3 rôzne stavy: Playing, Paused, Stopped, ktoré sú prepínané pomocou ovládacích tlačidiel.

Časovač je spustený pri prechode do stavu Playing a pri spustení má jeden parameter definujúci jeho rýchlosť (periódu) v milisekundách. Pri každom tiku je volaná funkcia `timerCallback()`, ktorá môže obsahovať príkazy vykonané pri každom tiku. V programe je funkcia využitá pre obnovenie grafického rozhrania (využité pri zobrazovaní pozície prehrávania), pre kontrolu zmeny stavu prehrávania a pre zobrazenie aktuálnej časti frekvenčného spektra. Pri kontrole zmeny stavu je kontrolovaná premenná označujúca zastavenie prehrávania (ktorú ovláda tlačidlo Stop) a následne prehrávanie pokračuje alebo je stav prehrávania zmenený na Stopped a časovač je zastavený.

6.5.4 Načítanie vzoriek

Pri prehrávaní je zvuk prehrávaný z výstupného buffera programu a jednotlivé vzorky je nutné postupne načítavať a posielať na zvukový výstup. Načítavanie a posielanie na výstup je ovládané funkciou `getNextAudioBlock()`, ktorá je implementovaná priamo v JUCE knižnici. V programe je táto funkcia explicitne upravená pre vlastné potreby. Jej volanie zabezpečuje tzv. JUCE Audio Thread, čo je súbor funkcií, ktoré vykonávajú procesy na pozadí programu spojené so spracovaním a prehrávaním zvuku. JUCE Audio Thread priamo komunikuje s Windows knižnicami, ktoré slúžia na ovládanie zvukového výstupu 6.5.5 (spracovanie zvuku určeného na prehranie a spravovanie zvukovej karty a výstupných zvukových zariadení).

Funkcia je volaná podľa potreby a načítava zvukovú stopu po blokoch vzoriek (načítavanie po jednotlivých vzorkách by bolo výpočetne náročné, kvôli potrebe volania v reálnom čase). Vo funkcií sú doplnené vlastné premenné a metódy ovládané prehrávacími prvkami, napr. premenné, ktoré ovládajú vykonávanie tejto funkcie, premenná pre určenie pozície prehrávania, ktorá je využitá pri vizualizácii a pretáčaní prehrávania a volanie metódy využívanej pri zobrazovaní frekvenčného spektra prehrávaného zvuku.

6.5.5 Komunikácia s výstupným zariadením

Pri prehrávaní zvukovej stopy musí byť program schopný zabezpečiť správnu komunikáciu s konečným zvukovým zariadením, aby bolo možné posielanie prehrávaného signálu na výstup.

Pri používaní JUCE frameworku sa o riešenie tohto problému starajú jednotlivé knižnice a ich metódy, ktoré sú na to priamo určené. Tieto knižnice komunikujú s programom a systémom a starajú sa o korektné spracovanie prehrávaného signálu (ako na obrázku 6.7). Hlavným komunikačným bodom programu pri prehrávaní signálu je funkcia `getNextAudioBlock()`.

Tento program je prispôsobený len pre systém Windows a preto je schopný komunikácie len s týmto operačným systémom. Metódy frameworku sú schopné zistiť, s ktorými systémovými knižnicami je potrebné komunikovať, pre zabezpečenie prehrávania a pre Windows je to napr. `win32_WASAPI`.

getNextAudioBlock()

Pri prehrávaní sa aktivuje, t.j. inicializuje sa vykonávanie jej volania a jej hlavnou funkciou je načítanie vzoriek z výstupného buffera. Funkcia je schopná komunikovať s knižnicou `audioSourcePlayer`. Funkcia má definovanú najvyššiu prioritu v rámci procesorového vlákna, na ktorom pracuje, čo znamená lepší výkon prehrávania, ale zároveň je nebezpečné vykonávať príliš zložité operácie v rámci tejto funkcie.

audioSourcePlayer

Riadi volanie programovej funkcie `getNextAuidioBlock()` podľa potreby. Pri volaní komunikuje zároveň s knižnicou `audioDeviceManager` a na základe informácií z danej knižnice ovláda volanie programovej funkcie. Okrem volania funkcie zároveň upravuje prehrávaný signál z danej funkcie, kvôli odstraňovaniu nežiaduceho efektu "praskania" pri spustení a zastavení prehrávania. Toto spracovanie vyhladzuje hrany signálu na začiatku a na konci blokov vzoriek, ktoré sú prijímané z programu.

audioDeviceManager

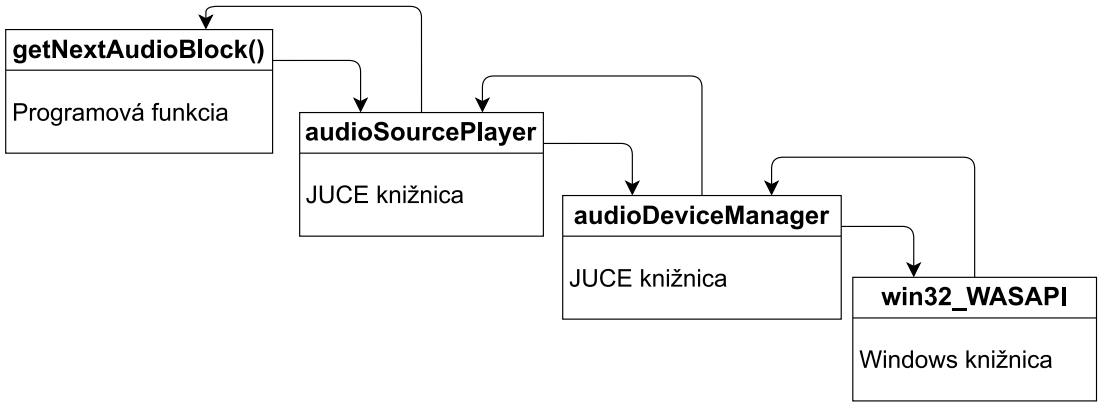
Obsahuje dôležité informácie o dostupných a používaných výstupných, ale aj vstupných zvukových zariadeniach a ich aktuálnom stave. Nastavenia spojené so zvukovými zariadeniami je možné uložiť a znovu načítať.

Umožňuje obojsmernú komunikáciu programu s Windows systémovou knižnicou, ktorá sa stará o zvukové zariadenia, t.j. posielanie signálu na výstupné zariadenie a prijímanie signálu zo vstupného zariadenia (napr. midi nástroje).

Udržiava neprerušovaný tok zvukového signálu na výstup / zo vstupu, ktorý je riadený volaniami z knižnice `audioSourcePlayer`.

win32_WASAPI

Poskytuje programu prístup k výstupnému / vstupnému audio zariadeniu a jeho stavu a k niektorým parametrom, ktoré sú potrebné pre komunikáciu s daným zariadením. Zároveň umožňuje posielať zvukový tok medzi audio zariadením a JUCE knižnicami, ktoré spracovávajú daný tok signálu.



Obrázek 6.7: Komunikácia programu s výstupom.

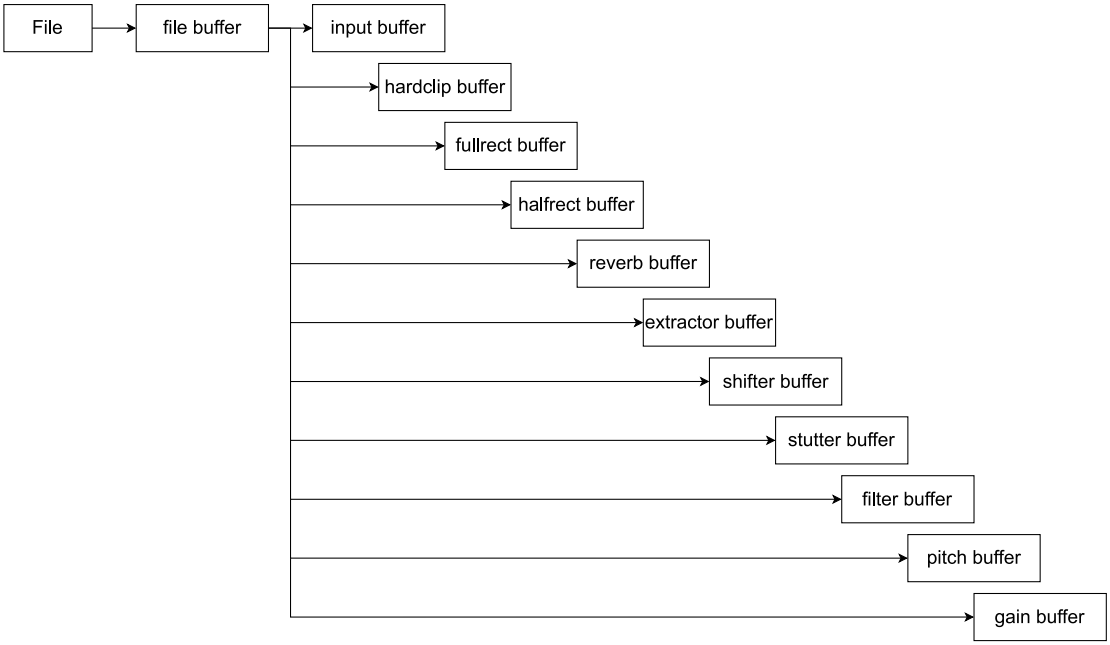
6.6 Príprava zvuku pre efekty

Po úspešnom načítaní zvukovej stopy z vybraného súboru musí byť zvuková stopa spracovaná a pripravená pre aplikovanie efektov. Spracovanie sa vykonáva okamžite po otvorení súboru a pozostáva z viacerých krokov, ktoré sú znázornené na obrázku 6.8.

Prvým krokom je nastavenie buffera (v programe `fileBuffer`), v ktorom bude udržiavaný zvukový signál zo súboru. Buffer je nastavený podľa veľkosti súboru a naplnený jeho obsahom.

Buffer je následne nastavený ako vstupný buffer pre všetky jednotlivé efekty. Toto nastavenie sa vykonáva kvôli spôsobu, akým efekty spracovávajú zvukový signál a akým sú navzájom prepojené, čo je popísané v sekcii 6.7.

Po nastavení buffera je získaná bitová hĺbka signálu, pre informovanie užívateľa a následne je povolené ovládanie parametrov všetkých efektov.



Obrázek 6.8: Načítanie zvukového signálu pre efekty.

6.7 Spracovací reťazec

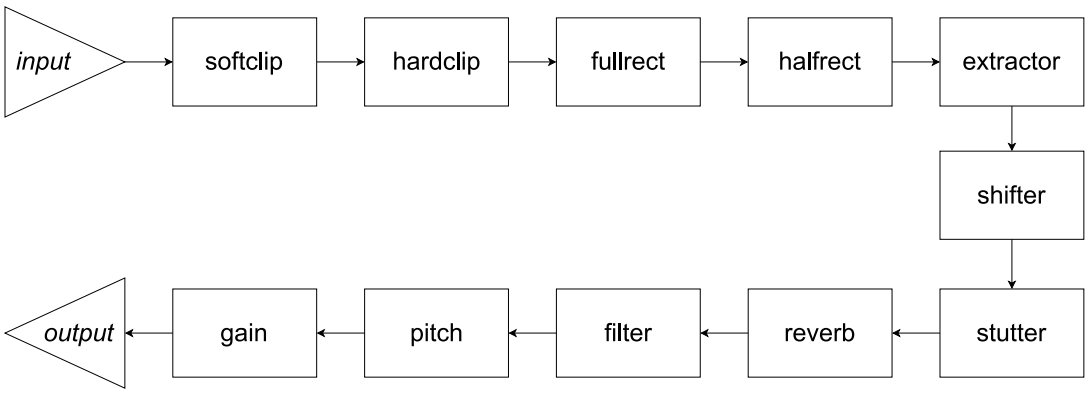
Pre správne spracovanie všetkých efektov a ich správnu aplikáciu je potrebné dosiahnuť niekoľko zásad. Pri spracovaní je potrebné, aby sa všetky efekty spracovávali v sérii a aby žiaden z nich nebol vynechaný. To znamená, že výstup každého jedného efektu musí byť zároveň vstupom ďalšieho efektu, ktorý za ním v sérii nasleduje. Sériové zapojenie efektov je znázornené na obrázku 6.9.

Zapojenie efektov však musí byť následne doplnené o ďalšiu vlastnosť kvôli skutočnosti, že pri spracovaní signálu nemusia byť aktívne všetky efekty, ale len tie, ktoré si užívateľ vyberie. Bez ošetrenia tejto skutočnosti v programe vždy nastane situácia, kedy sa kvôli sériovému zapojeniu efektov spracovací reťazec rozpojí v miestach, kde sú efekty vypnuté.

Tento problém je vyriešený tým, že každý efekt je schopný posielať vstupný signál na výstup aj pri vypnutom (“dry”) stave. Rozdiel od zapnutého stavu je len v tom, že v tomto prípade sa signál vôbec nespracováva, t.j. výstup je identický vstupnému signálu.

Zo sériového zapojenia vyplýva ďalšia vlastnosť spracovacieho reťazca, a to nutnosť spracovávať len niektoré efekty zo série. Ak je počas úpravy signálu aplikovaný a nastavovaný určitý efekt, tak pre aktualizáciu výsledného signálu v skutočnosti nie je potrebné spracovať všetky aktívne efekty v sérii. Potrebu spracovania má len daný efekt a všetky aktívne efekty, ktoré v sérii nasledujú. V programe má každý efekt svoj poradový index a pri nastavovaní určitého efektu sa spracovávajú len tie s rovnakým a vyšším indexom. Táto vlastnosť umožňuje optimalizáciu programu vďaka zníženiu výpočtového času a záťaže na procesor.

Daná optimalizácia spracovania však vytvára jeden nežiaduci problém so vstupnými buffermi jednotlivých efektov. Ak sa spracovávajú len určité efekty v sérii, tak pre úplne prvé spracovanie ľubovoľného z efektov hneď po načítaní súboru je potrebné nastaviť daný súbor ako vstup pre každý z efektov, pretože inak by bol po bezprostrednom načítaní funkčný len prvý efekt zo série. Riešenie je popísané na obrázku 6.8. Načítanie súboru do každého efektu sa vykonáva len raz po načítaní súboru, čiže výkon programu nie je týmto ovplyvnený.



Obrázek 6.9: Spracovací reťazec pre efekty

Spracovací reťazec je implementovaný vo funkcii `processAllEffects(index)`, ktorá spracováva efekty v poradí a spracovávanie začína od efektu určeného parametrom `index`. Po úspešnom spracovaní funkcia zavolá vykresľovanie zvukovej vlny signálu a výsledok je zobrazený v okne vizualizácie zvukovej vlny.

6.7.1 Vyvolanie spracovania

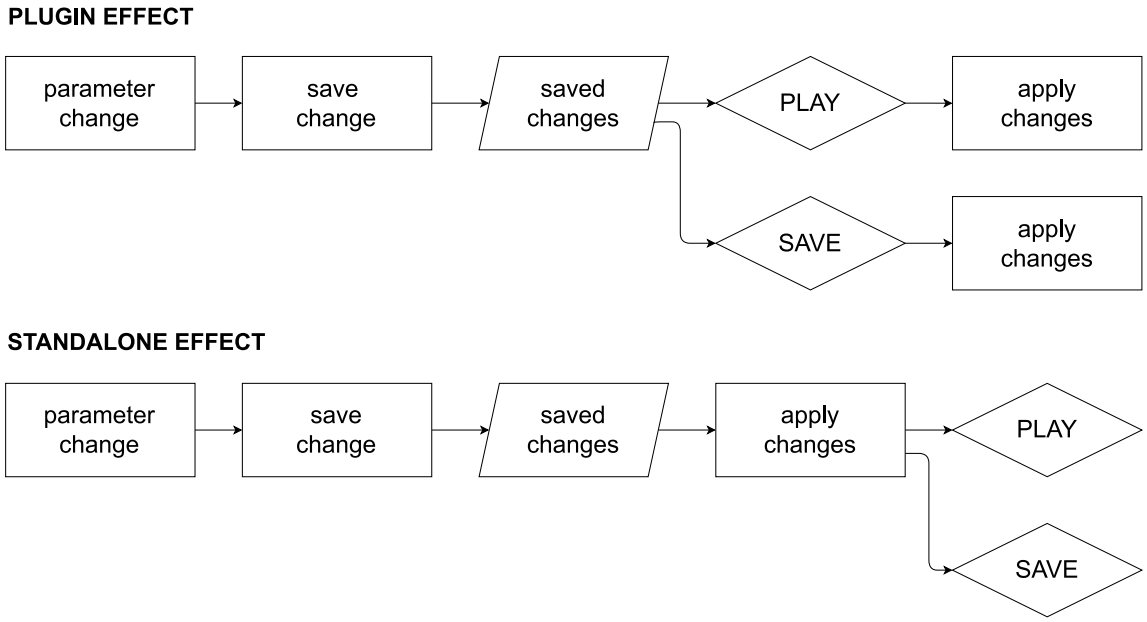
Pri spracovávaní je dôležité, za akých okolností a v ktorých prípadoch sa má vykonať spracovanie série efektov, čiže aká akcia má vyvolať spracovanie a aplikáciu efektov.

V bežných efektoch, ktoré sa používajú v plugin verziách a sú používané v softvéri na tvorbu hudby (DAW) sa spracovávanie vykonáva len pri prehrávaní, alebo ukladaní výslednej zvukovej stopy do súboru. Tento spôsob je používaný hlavne z dôvodu rýchlosti a responzivity programu. Tento typ softvéru slúži nielen na úpravu krátkych zvukových stôp, ktorých dĺžka sa pohybuje od niekoľkých jednotiek do desiatok sekúnd, ale aj na úpravu celých skladieb, ktoré sa pohybujú v rozsahu jednotiek minút. Vzhľadom na množstvo efek-

tov a syntetizátorov používaných pri tvorbe skladieb by bolo spracovávanie celej skladby okamžite pri zmene parametrov každého efektu časovo náročné a nepraktické.

V našom programe je však spracovanie spracovacieho reťazca vyvolávané pri každej zmene parametra každého z efektov. Tento spôsob je dostatočný hlavne kvôli tomu, že sa jedná o samostatný program, ktorý nie je možné použiť ako plugin v inom softvéri, a je určený len pre spracovanie krátkych zvukových stôp. To síce znamená, že dĺžka spracovaného signálu a zároveň aj množstvo efektov je oveľa nižšie, ale spracovanie efektov a aj ostatných prvkov programu zaberá menej výpočtového času. Rozdiel medzi aplikáciou efektov v DAW softvéri a v tomto programe je vidieť na obrázku 6.10.

Ďalším dôvodom okamžitého spracovávania je okamžité zobrazovanie celej zvukovej vlny po aplikácii efektov. Ak by bol signál spracovávaný takým spôsobom, ako v DAW, čiže len pri prehrávaní alebo ukladaní do výsledného súboru, tak by to znamenalo, že efekty by spracovávali signál po blokoch a zobrazovanie vlny by bolo možné len pre tieto aktuálne spracovávané bloky, alebo až po uložení celého súboru.



Obrázek 6.10: Rozdiel v aplikácii zmien v DAW / v našom programe

6.7.2 Spracovacia cesta efektu

Vyvolávanie spracovania efektov a následná spracovacia cesta efektu, t.j. postupnosť metód vykonávaných pri spracovaní efektov (popísané na obrázku 6.11) sú riadené dvomi udalosťami.

Prvou udalosťou je klik na tlačidlo, ktoré je priradené danému efektu. Kliknutie vyvolá metódu `process<effect>ButtonClicked`, ktorá nastaví premennú viazanú na stav tlačidla (typu `bool`) na `true` alebo `false`. Premenná je nastavená podľa aktuálneho stavu efektu, čiže efekt je podľa stavu prepnutý na iný stav. Po zapnutí / vypnutí efektu je následne volaná funkcia `processAllEffects()`.

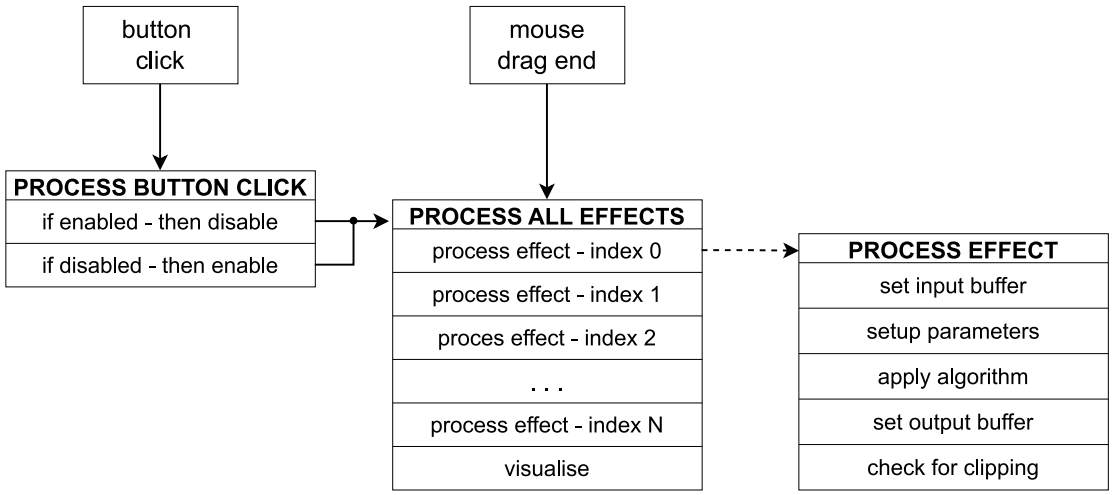
Táto funkcia pri spracovaní každého efektu volá funkciu `process<effect>SliderChange()`, ktorá ďalej využíva metódy samotných efektov pre spracovanie zvukového signálu. Po úspeš-

nom spracovaní jedného efektu je spracovaný ďalší, až kým sa metóda `processAllEffects()` dostane na koniec spracovania.

Druhou udalosťou je interakcia myši s prvkom posuvníka efektu. Myš obsahuje vo frameworku viacero metód, ktoré program automaticky volá, pokiaľ systém zaregistruje určitú činnosť myši a v ktorých je možné definovať vlastné akcie na vykonanie pri danej činnosti (okrem implicitne definovaných). Metódy slúžia pre spracovanie činností, ako napr. stlačenie tlačidla myši, pohyb, stlačenie a potiahnutie, a podobne. Jedna konkrétna činnosť, ktorá je využívaná pre spracovanie efektov je `onDragEnd()`.

onDragEnd()

Táto činnosť úzko súvisí s činnosťou `onDragStart()`, ktorá je vyvolávaná, pokiaľ program zaregistruje stlačenie tlačidla myši a následný posun myši. Činnosť `onDragEnd()` označuje uvoľnenie tlačidla myši počas posunu. K tejto činnosti je možné priradiť jednoduchú lambda metódu, ktorá môže obsahovať vlastný príkaz na vykonanie. V tomto prípade je k činnosti priradené volanie funkcie `processAllEffects()`, čo znamená, že pri každom skončení posúvania parametru sa vykonajú procesy vedúce k zmene parametra a spracovaniu efektov (ako je znázornené na obrázku 6.11).



Obrázek 6.11: Spracovacia cesta efektov

6.8 Dodatočné súčasti

6.8.1 Gain - zmena amplitúdy

Zmena amplitúdy upravuje amplitúdu zdrojovej stopy. Amplitúda sa dá týmto nastavením znížiť alebo zvýšiť v rozsahu od -40dB do 40dB. Zmena amplitúdy sa aplikuje na konci spracovania signálu a je využiteľná pri určitých efektoch, ktoré môžu znižovať amplitúdu (distortion, filter), alebo pri efektoch, ktoré môžu zvyšovať amplitúdu (reverb).

Využívaná je funkcia `applyGain()`, ktorá aplikuje Gain priamo na celý buffer so zvukovým signálom. Parametrom funkcie je koeficient, ktorým sa násobia hodnoty jednotlivých vzoriek. Kvôli tomu je pri spracovaní potrebné previesť hodnotu z posuvníka efektu, ktorý má logaritmický rozsah v dB na ekvivalentnú lineárnu hodnotu koeficientu.

6.8.2 Bitová hĺbka

Nastavenie bitovej hĺbky upravuje bitovú hĺbku výstupného súboru na uloženie. Pri načítaní signálu je posuvník pre nastavenie bitovej hĺbky nastavený na hodnotu, ktorú má načítaná zvuková stopa a túto hodnotu je následne možné zmeniť. Táto zmena sa aplikuje až pri uložení stopy do súboru. Dostupné hodnoty sú 16, 24 a 32 bitov.

6.8.3 Klávesové skratky

Pre zjednodušenie používania zahŕňajú všetky hudobné aplikácie aj možnosť ovládania pomocou klávesových skratiek. Bežne sa skratky využívajú pre otvorenie a uloženie súborov, ale aj pre ovládanie prehrávania, či iné operácie so zvukovými stopami.

V programe je implementovaných niekoľko rôznych klávesových skratiek a to:

- `O` alebo `o` - otvorenie súboru
- `S` alebo `s` - uloženie súboru
- `Q` alebo `q` - spustenie / pozastavenie prehrávania
- `W` alebo `w` - zastavenie prehrávania

Všetky klávesové skratky sú spracovávané funkciou `keyPressed()`, ktorá na základe vlastností (typu) stlačeného klávesu vykoná príslušný príkaz. Klávesové skratky pracujú na princípe vyvolania stlačenia určitého tlačidla pomocou funkcie `triggerClick()`. Pri stlačení ľubovoľného z implementovaných klávesov sa vyvolá stlačenie tlačidla, ktoré súvisí s daným klávesom, napr. pre otvorenie súboru je to tlačidlo `openButton`. Stlačenie je simulované metódami z JUCE knižnice pre správu klávesnice a vo výsledku je identické fyzickému stlačeniu daného klávesu.

6.8.4 Reset nastavení

Resetovanie nastavení pre všetky efekty je súčasť implementovaná na základe odozvy z testovania nášho programu. Táto funkcionalita slúži najmä pre zrýchlenie práce s programom pri nastavovaní hodnôt efektov.

Reset je implementovaný v jednoduchej funkcii `resetEffectButtonClicked()`. Funckia nastaví hodnoty efektov na ich pôvodné hodnoty a vypne všetky efekty, čím ich odstráni z upravovaného súboru.

6.8.5 Nápoveda

Nápoveda je implementovaná kvôli návrhu z užívateľského testovania. Slúži pre zrýchlenie a zjednodušenie orientácie v programe, a to hlavne pri prvom použití nášho programu. Obsahuje krátky popis určitých prvkov programu.

Nápoveda je implementovaná pomocou triedy `Tooltip` z JUCE frameworku. Je zobrazená vo forme plávajúceho okna, ktoré je umiestnené pri kurzore myši. Okno sa zobrazuje len pre určité prvky, pre ktoré je daná nápoveda nastavená. Nápovedu je možné prepínať pomocou tlačidla `Tooltips`.

6.8.6 Drag and Drop

Drag and Drop (ďalej DnD) je funkcionalita určená pre načítanie súboru. DnD umožňuje načítať súbor cez pretiahnutie daného súboru z priečinka na ľubovoľné miesto v okne. DnD je implementovaná vo funkcii `filesDropped()`. Táto funkcia zistí názov súboru a následne predá súbor funkcii, ktorá spracováva aj súbory otvorené cez tlačidlo `Open`. DnD slúži na praktickú manipuláciu so súbormi, pretože niektorí užívatelia môžu byť zvyknutí načítavať súbory práve týmto spôsobom.

6.9 Štruktúra jednotlivých efektov

Samotné algoritmy a spracovanie efektov sa v rámci projektu nachádzajú samostatne vo svojich triedach. Hlavičkový súbor každého efektu obsahuje všetky potrebné definície funkcií a parametrov využívaných v implementácií efektu a jeho algoritmov. Zdrojový súbor triedy obsahuje vždy predspracovanie zvukového signálu pre vstup do efektového algoritmu a následne aj samotný algoritmus efektu.

6.9.1 Predspracovanie efektu

Vstupom každého efektu je tzv. `inputBuffer`, ktorý obsahuje signál pripravený na spracovanie. Avšak pre dôvody uvedené v sekcii 6.7 je potrebné vytvoriť kópiu tohto buffer-u, tzv. `outputBuffer`, ktorý je identickou kópiou vstupného buffer-u. Táto kópia zároveň slúži aj pre možnosť vracať späť zmeny vykonané v efekte. Je potrebné dosiahnuť, aby sa efekt aplikoval vždy na originálny vstupný signál a nie na signál spracovaný predchádzajúcim aplikovaním daného efektu. Bez vytvárania kópie by sa spracovaný signál vždy uložil do vstupného buffer-u a bol by použitý znovu ako vstup pre daný efekt.

Po nastavení buffer-u sú následne spracované a uložené parametre predané z hlavnej časti programu. V niektorých efektoch je nutné aj inicializovať a nastaviť pomocné súčasti, ako napr. v efekte Stutter je potrebné nastaviť pomocný Chorus efekt ešte pred vykonaním samotného algoritmu efektu.

Po úspešnom predspracovaní nasleduje vykonanie hlavného algoritmu, ktorý aplikuje efekt na zvukový signál.

6.9.2 Algoritmus efektu

Po predspracovaní je signál pripravený prejsť úpravou algoritmom efektu. Algoritmus realizujúci efekt je vo funkcii `add<Effect>()` (napr. pre efekt Reverb je to `addReverb()`), ktorá po úspešnom spracovaní signálu vždy navráti výstupný buffer s výsledným signálom.

Vzhľadom na to, že pri spracovaní efektu sa vytvára kópia vstupného buffera, je tento buffer v prípade vypnutia efektu nastavený na veľkosť 1 vzorky (ale nie úplne odstránený), čo slúži na ušetrenie využívanej pamäte zariadenia.

6.10 Presety

Presety sú nepovinnou, ale vhodnou súčasťou hudobných aplikácií s efektami, v ktorých je možné meniť rôzne parametre efektov. Presety sú užitočné hlavne pre zložitejšie aplikácie, ako napr. tento multiefektový program. Presety uľahčujú užívateľom prácu s programom, a to urýchlením orientácie pri prvých pokusoch o použitie programu, ale zároveň slúžia aj na ukážku výsledných efektov, ktoré sa dajú dosiahnuť s daným programom.

Presety obsahujú uložené nastavenia jednotlivých parametrov programu (hlavne parametrov samotných efektov v programe) v súboroch v určitom textovom formáte, ktoré je možné ľahko vytvárať a upravovať.

Jednou z najjednoduchších foriem ukladania efektov je použitie štrukturovaného textového formátu, ako napr. XML. Tento formát umožňuje ukladať parametre vo vlastnom stanovenom formáte a pri načítaní presetu je možné získať tieto parametre, keďže formát je vopred známy.

V programe je použitie XML výhodne aj kvôli možnosti využitia metód z triedy JUCE frameworku s názvom `XmlElement`. Trieda obsahuje všetky potrebné metódy pre ukladanie parametrov vo forme XML elementov a vytváranie konečného súboru z týchto elementov a zároveň aj metódy pre načítanie elementov z presetu a získanie hodnôt pre nastavenie parametrov. Práca s XML súbormi je v programe implementovaná v dvoch funkciách, a to `saveXmlElements()` pre ukladanie presetov a `parseXmlElements()` pre načítanie presetov.

Formát presetu pozostáva z nasledovných častí:

- Hlavička XML súboru - ktorá je povinná pre XML súbory a určuje najmä kódovanie súboru.
- Elementy - obsahujú dáta súboru a sú usporiadané hierarchicky v stromovej štruktúre (jednotlivé elementy sú "listy" stromovej štruktúry).

Ukážka súboru (6.12) znázorňuje formát, ktorý je použitý v programe.

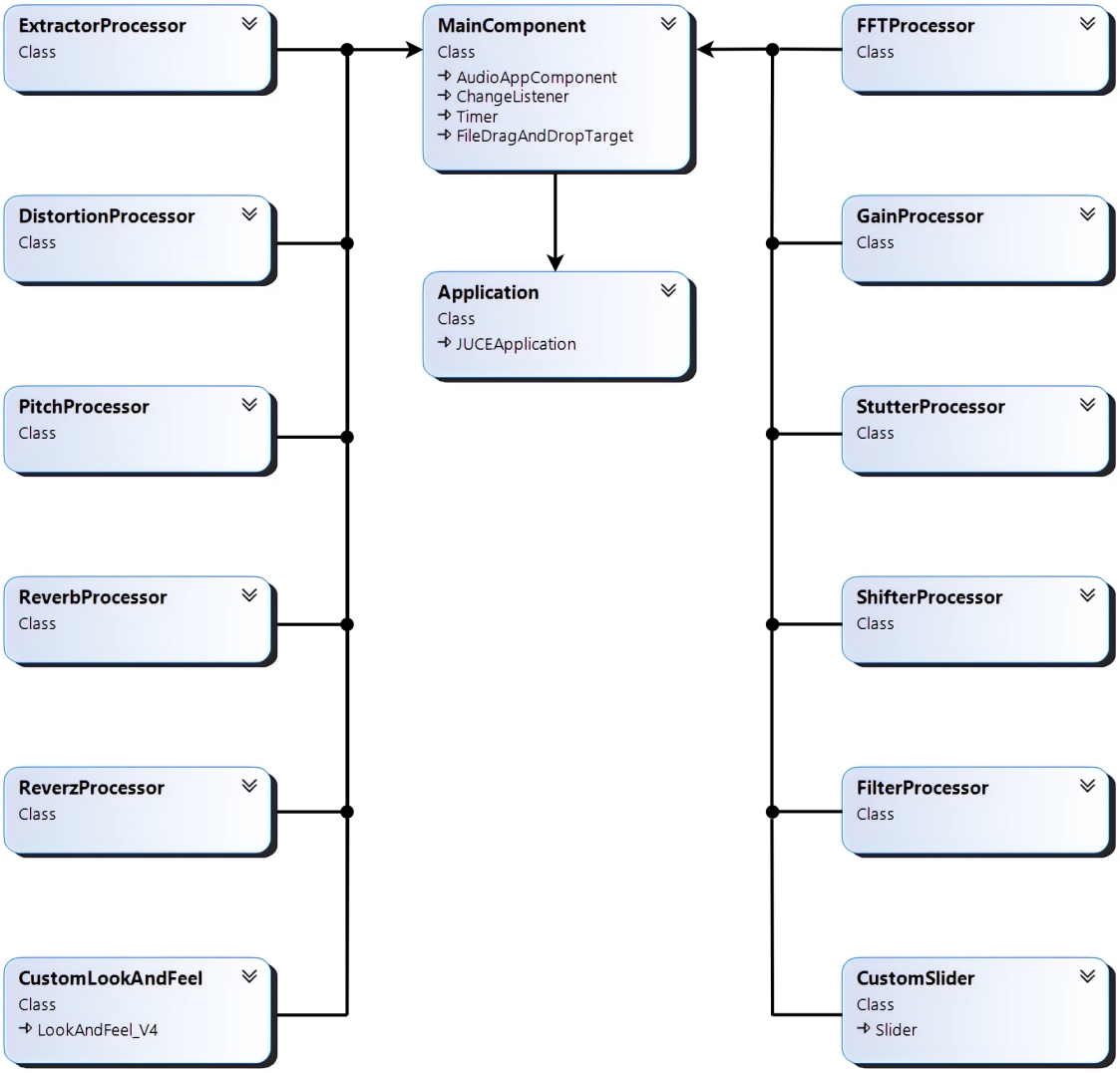
```
<?xml version="1.0" encoding="UTF-8"?>

<Parameters>
  <Gain Enabled="1.0" Value="-3.199999999999996"/>
  <Pitch Enabled="0.0" Value="1.0"/>
  <LPF Enabled="1.0" Cutoff="20000.0" Q="1.0"/>
  <HPF Enabled="1.0" Cutoff="201.0" Q="1.0"/>
  <BPF Enabled="0.0" Cutoff="671.0" Q="1.0"/>
  <Reverb Enabled="1.0" Balance="0.25" Size="0.5" Width="1.0" Damp="0.75"/>
  <SCD Enabled="1.0" Threshold="0.75"/>
  <HCD Enabled="0.0" Threshold="0.0"/>
  <FRD Enabled="0.0"/>
  <HRD Enabled="0.0"/>
  <Extractor Enabled="0.0" Intensity="0.0" Width="0.0"/>
  <Reverz Enabled="1.0" Amount="16.0" Skew="0.0"/>
  <Stutter Enabled="1.0" Amount="16.0" Chorus="10.0" Delay="50.0"/>
  <Shifter Enabled="1.0" Amount="32.0" Tone="2.0"/>
</Parameters>
```

Obrázek 6.12: Ukážka XML presetu.

Niektoré z vytvorených užívateľských presetov sú uvedené v prílohe B.

6.11 Diagram tried

Na obrázku 6.13 je uvedený diagram tried projektu. Diagram obsahuje len prepojenie vlastných definovaných tried, bez tried zo systémových knižníc alebo tried JUCE frameworku.



Obrázek 6.13: Diagram tried nášho programu.

Kapitola 7

Užívateľské rozhranie

Grafické rozhranie je dôležitá súčasť hudobného softvéru ovplyvňujúca používanie aplikácie. Zásadným spôsobom ovplyvňuje intuivitu, náročnosť používania programu a celkový konečný dojem z programu.

Pri návrhu a implementácií rozhrania je nesmierne dôležité vedieť, aké rozhrania sú využívané v existujúcich programoch a ako vyzerá dizajn jednotlivých prvkov grafického rozhrania.

Preto bol pred návrhom grafického rozhrania vykonaný prieskum, ktorého cieľom bolo preskúmať vzhľad existujúcich hudobných programov a na ich základe určiť vlastný dizajn nášho programu.

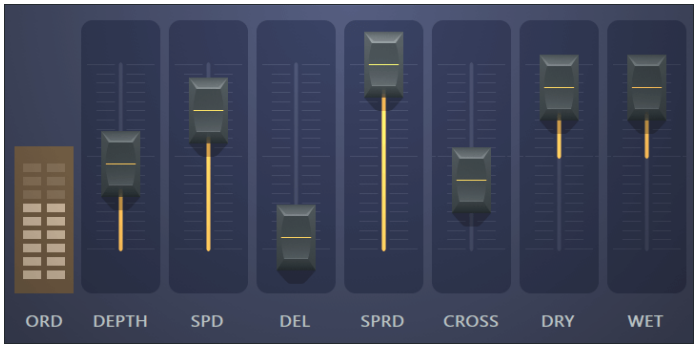
7.1 Prieskum grafických prvkov

Prieskum pozostával z vyhľadania často používaných programov a porovnania ich grafických rozhraní. Pri porovnávaní bol braný ohľad na ovládanie jednotlivých parametrov efektov, rozloženie ovládacích prvkov, vizualizačné funkcie a vzhľad doplnkových funkcií.

Predmetom prieskumu boli medzi inými aj nasledujúce programy:

Obrázek 7.1: Program Fast Dist od Image-Line. Prevzaté z [12].

Fast Dist (obrázok 7.1) je jednoduchý efekt obsahujúci 4 parametre a jeden grafický prvok. Pre náš program je zaujímavý dizajn ovládania daných parametrov. Ovládanie je realizované cez tzv. “knob”, čo je otočný posuvník. Knob je grafický prvok prevzatý z hardvérových efektov, v ktorých slúži na ovládanie potenciometra regulujúceho napätie. Knob má v rámci softvéru aj hardvéru výhodu vo svojom kruhovom tvare vďaka čomu zaberá menší priestor, než ostatné typy posuvníkov. V našom programe je knob použitý na ovládanie všetkých parametrov zvukových efektov.



Obrázek 7.2: Program Flangus od Image-Line. Prevzaté z [13].

Flangus (obrázok 7.2) je efekt, ktorý namiesto knobu využíva na ovládanie parametrov normálny posuvník. Posuvník je využívaný približne v rovnakej miere, ako knob a spolu sú to dva najrozšírenejšie ovládacie grafické prvky v hudobných efektoch. V našom programe je využitý pre nastavenie bitovej hĺbky a rozlíšenia frekvenčného spektra signálu.

Obrázek 7.3: Program Waveform Free od Tracktion. Prevzaté z [26].

Program Waveform Free (obrázok 7.3) je DAW softvér, ktorý obsahuje množstvo rôznych efektov a vizualizačných prvkov. Tento program bol skúmaný hlavne kvôli vizualizácii zvukovej vlny a frekvenčného spektra. Obidva vizualizačné prvky sú v danom programe implementované ako jednoduché okná. Pre vykreslenie používajú jednoduché krivky, ktoré sú tvarom prispôsobené požadovanému výsledku. Tento štýl vizualizácie (ktorý je bežný vo väčšine DAW) bol zvolený ako inšpirácia pre vizualizácie v našom programe.

Zvyšné grafické prvky, ako napr. tlačidlá, textové popisy a textové polia boli navrhnuté podľa vlastného dizajnu.

7.2 Návrh vlastných prvkov

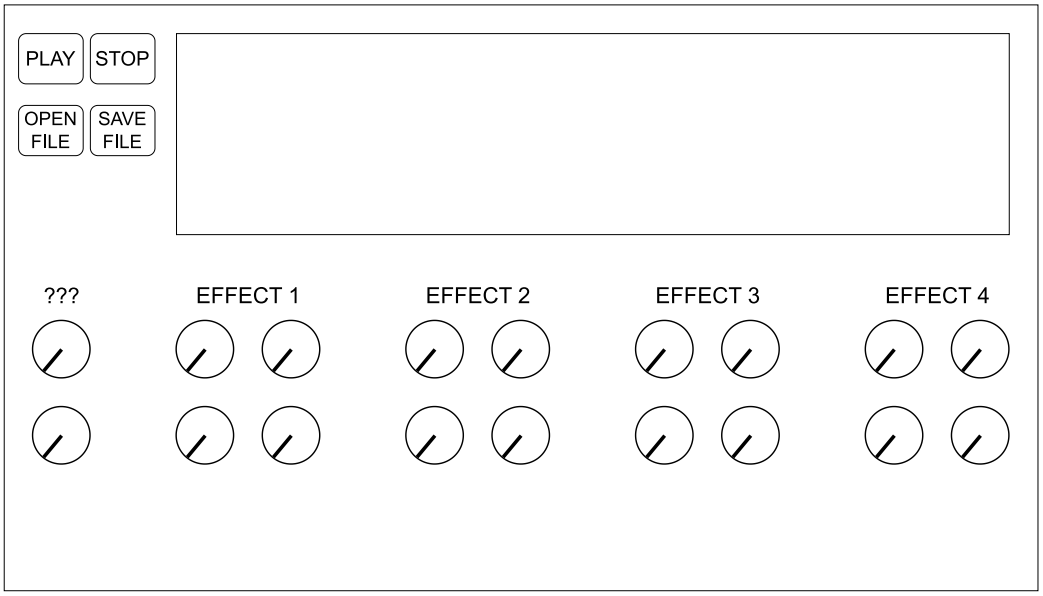
Po vyhodnotení prieskumu a určení približného dizajnu pre grafické súčasti bolo vytvorených viacero návrhov nášho programu v aplikácii `draw.io`. Tieto návrhy slúžili len ako približný návod pre rozmiestnenie grafických prvkov vo vývojovom prostredí, z čoho vyplýva aj ich jednoduchosť.

Pri návrhu bolo dôležité určiť vhodné rozmery pre každý prvok a vhodné umiestnenie v rámci okna programu. V rámci umiestnenia boli dodržiavané niektoré zásady, ako napríklad:

- Podobné prvky by mali byť rozčlenené do skupín a umiestnené vo vzájomnej blízkosti
- Medzi skupinami prvkov by malo byť viac voľného miesta, než medzi prvkami v rámci skupiny
- Prvky vedľajších funkcií programu by mali byť umiestnené na ľavej / pravej strane
- Efekty by mali byť pod oknami s vizualizáciou

Tieto zásady boli skutočne dodržiavané až v neskorších fázach vývoja nášho programu, v ktorých program obsahoval väčšie množstvo grafických prvkov.

Na obrázku 7.4 je uvedený prvý návrh grafického rozhrania nášho programu.

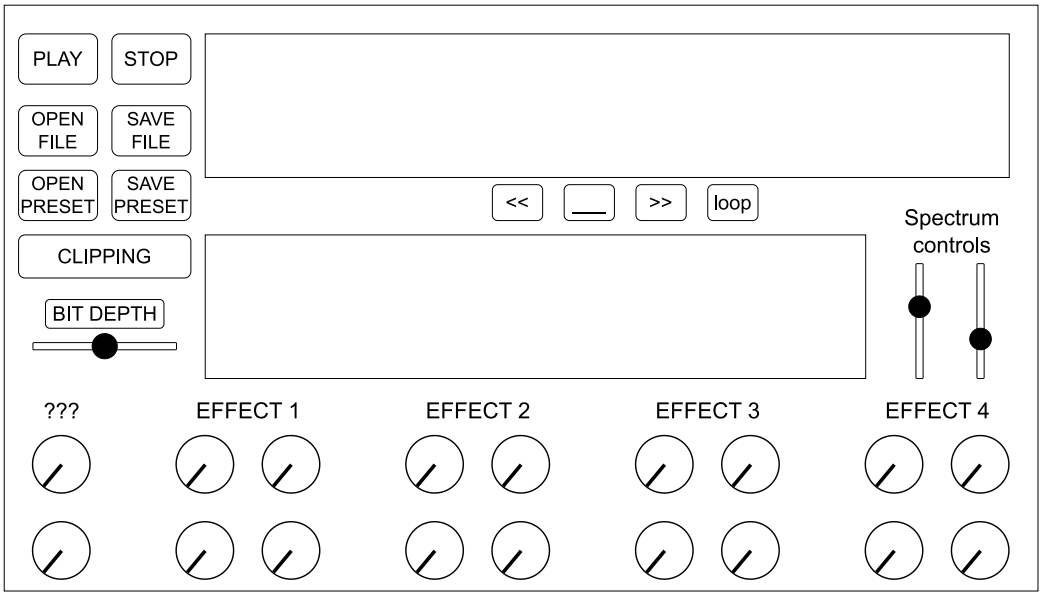


Obrázek 7.4: Prvý návrh grafického rozhrania.

Prvý návrh obsahuje len základné ovládanie prehrávania, súborov, niekoľkých efektov a vizualizáciu zvukovej vlny. V tejto fáze sme sa zameriavali najmä na funkcionalitu, čiže návrh predstavoval len "vodítko" pre približnú predstavu vzhľadu nášho programu.

Návrh uvedený na obrázku 7.5 je jeden z posledných návrhov vytvorených v konečných fázach vývoja a obsahuje v podstate takmer všetky grafické prvky. Z návrhu vidieť, že jednotlivé efekty sú rozdelené do skupín a umiestnené pod dve vizualizačné okná. Ovládacie prvky pre súbory, prehrávanie a vedľajšie funkcie sú na ľavej strane. Dodatočné ovládanie pre vizualizačné okna je umiestnené čo najbližšie k oknám, aby užívateľ videl, že sa jedná o ovládanie daných okien.

Od daného návrhu v programe pribudli len dve tlačidlá pre ďalšie doplnkové funkcie, ale vzhľad a umiestnenie zvyšných prvkov neboli vo finálnej verzii zmenené. Výsledný dizajn spĺňa naše určené zásady a prvky sú podobné s grafickými prvkami v existujúcich programoch. Vďaka tomu je orientácia menej náročná, než pri kompletne vlastnom dizajne.



Obrázek 7.5: Pokročilý návrh grafického rozhrania.

7.3 Implementácia rozhrania

Grafické rozhranie je implementované v hlavnej triede programu `MainComponent`. Každý grafický prvok je inštanciou konkrétnej triedy z JUCE frameworku, ktorá obsahuje implementáciu funkcionality daného prvku.

V našom programe sa vyskytuje viacero typov prvkov, a to nasledovné:

- Label - textový štítok
- Slider - posuvník
- TextButton - tlačidlo s textom
- TextEditor - textový vstup

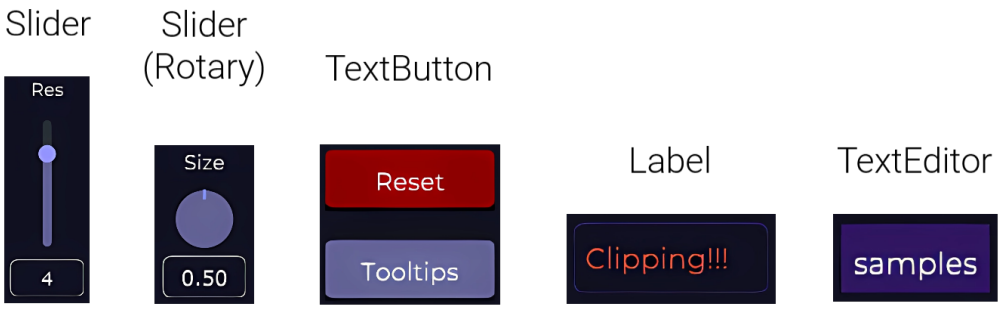
Grafický dizajn každého prvku je implicitne implementovaný v triede frameworku s názvom `LookAndFeel`. Táto trieda definuje základný vzhľad grafických prvkov a obsahuje metódy pre vlastnú zmenu určitých vlastností. Jednotlivé metódy majú určitý preddefinovaný zdrojový kód, avšak k dispozícii je možnosť redefinície a rozšírenia daných metód pomocou identifikátora `override`. V programe je táto možnosť využitá a preto náš program obsahuje vlastnú triedu `CustomLookAndFeel`. Táto trieda upravuje metódy niektorých grafických prvkov, pri ktorých nebolo možné dosiahnuť požadovaný vzhľad pomocou zmien nastavenia bežnými spôsobmi.

Trieda `CustomLookAndFeel` je dôležitá pre:

- vlastné otočné posuvníky
- zaoblené rohy tlačidiel a štítkov
- vlastné písmo

- odsadenie písma v tlačidle
- animácia pri kliknutí na tlačidlo

Na obrázku 7.6 je znázornený dizajn všetkých typov ovládacích prvkov.



Obrázek 7.6: Dizajn ovládacích prvkov.

7.3.1 Geometrické útvary

Pre vykresľovanie jednoduchých grafických útvarov, ako napr. čiary a obdĺžniky existuje v JUCE frameworku funkcia `paint()`. Funkcia používa grafický objekt, tzv. `Graphics`, ktorý poskytuje všetky potrebné prostriedky pre vykreslenie základných geometrických útvarov.

Funkcia `paint()` sa vyvoláva podľa programového časovača (sekcia 6.5.3) a je možné ju vyvolať aj manuálne pomocou funkcie `repaint()`. V našom programe je použitá pre vykreslenie vizualizačných okien, ich obsahu a pre jednoduché ohraničenia jednotlivých efektov. Pre vykreslenie ľubovoľného útvaru je vždy potrebné najprv definovať aspoň rozmery a farbu útvaru (mimo iných vlastností). Útvary je možné definovať aj v iných funkciách, ale iba vo funkcii `paint()` môže byť vykonané konečné vykreslenie daných útvarov.

Kapitola 8

Testovanie

8.1 Vlastné testovanie

Testovanie bolo vykonávané vo vývojovom prostredí Microsoft Visual Studio Community 2019, v ktorom bol vyvíjaný tento program, pretože obsahuje dostatočné nástroje pre vlastné testovanie a sledovanie činnosti programu. Testovanie prebiehalo s pomocou tzv. Debug módu, ktorý umožňuje sledovať jednotlivé funkcie a premenné v skompilovanej a spustenej aplikácii.

Testovanie prebiehalo s určeným scenárom, ktorý obsahoval viacero krokov:

1. Kompilácia programu
2. Spustenie programu
3. Otvorenie súboru
4. Prehrávanie súboru
5. Nastavenie parametrov efektu
6. Prehrávanie upraveného súboru
7. Uloženie upraveného súboru
8. Opakuj krok 5

Tento scenár nebol používaný od začiatku testovania, ale osvedčil sa ako najlepší z použitých možností. Testovanie v daných krokoch bolo opakované do otestovania všetkých implementovaných efektov, alebo do prerušenia chodu programu chybou spôsobenou jedným z týchto efektov.

Testovanie po jednotlivých efektoch malo výhodu v obmedzení pôvodu chyby na menšiu časť programu a vďaka tomu mohla byť daná chyba rýchlejšie identifikovateľná a opravená. Pri určení chyby bolo využívané vloženie tzv. breakpointu do zdrojového kódu na predpokladané miesta, kde zrejme dochádza k chybám v spracovaní.

Breakpoint umožňuje pozastaviť program na určitom kroku spracovania a preskúmať hodnoty premenných v danom mieste. Zároveň, ak program narazí na tento bod, tak je možné spustiť "krokovanie" programu. Krokovanie umožňuje zistiť, ktoré príkazy sú vykonávané a ktoré funkcie daného programu sú volané.

Pri testovaní sa vyskytovali najčastejšie problémy s nesprávnou inicializáciou niektorých súčastí programu, v ktorých boli využívané objekty z frameworku (napr. nesprávna inicializácia efektu Chorus, Reverb alebo Gain) a problémy s neoprávneným prístupom do pamäte (hlavne pri vlastných glitch efektoch), kedy index spracovávanej vzorky bol nesprávne nastavený mimo spracovávaného signálu. Tieto problémy boli vyriešené hlavne vďaka krokovaniu a sledovaniu hodnôt veľkosti buffera so signálom a indexov spracovaných vzoriek.

Niektoré príklady rôznych typov riešených problémov:

- Trieda `StutterProcessor`: nedostatočná inicializácia parametrov triedy `Chorus`.
- Trieda `ReverzProcessor`: prístup mimo vyhradenú pamäť pri použití vyhladzovacej funkcie.
- Trieda `PitchProcessor`: prístup mimo vyhradenú pamäť kvôli nesprávnej zmene veľkosti buffera pri zmene dĺžky signálu.
- Trieda `MainComponent`: únik pamäte pri používaní dynamicky alokovaných inštancií triedy `XmlElement`.

Pri testovaní kvality zvukového výstupu boli použité rôzne zvukové stopy, pričom najdôležitejšie bolo dosiahnuť, aby bolo použitých čo najviac rôznych typov zvukov. Medzi jednotlivými typmi boli využívané nasledovné:

- Basová linka
- Pad
- Lead
- Vokály
- Bicie (prevažne elektronické)
- Perkusia
- Gitara
- Jednoduchá sínusoida
- Biely šum

Pri použití rôznych zvukov bolo zaistené presnejšie zistenie problémov s kvalitou výstupu, pretože každý efekt môže mať iný vplyv na rôzne typy zvukov. Ďalej pri vývoji vlastných glitch efektov bolo zistené vhodné využitie týchto efektov, t.j na ktoré zvuky je vhodné použiť dané glitch efekty a pri ktorých zvukoch je výsledok skôr nedostačujúci a nevyhovujúci.

Ďalším spôsobom testovania bolo sledovanie využívania procesorového výkonu a dočasnej pamäte. Tento typ testovania bol používaný až po úspešnom otestovaní funkčnosti jednotlivých efektov a doplnkových súčastí programu. Sledovanie daných dvoch vlastností bolo dôležité najmä pre kontrolu výkonu programu a nájdenie slabých miest v programe, kde dochádza k operáciám, ktoré vedú ku nadmernému a v určitých prípadoch aj zbytočnému zníženiu výkonu alebo zvýšeniu množstva používanej pamäte.

Pri danom testovaní boli nájdené viaceré miesta v programe, ktoré nadmerne zaťažovali systém. V niektorých miestach bola táto záťaž prirodzená a vyplývala z princípu danej súčasti programu, avšak v iných miestach bolo možné previesť optimalizácie daných súčastí a dosiahnuť vyššiu efektivitu programu. Konkrétne slabé miesta sa nachádzali hlavne pri spracovávaní samotných algoritmov efektov, kde častokrát dochádzalo k vykonávaniu zbytočných operácií a k využívaniu vyššieho množstva pamäte, než bolo potrebné. Pri testovaní výkonu a pamäte boli takisto využité prostriedky dostupné v prostredí MS Visual Studio a to konkrétne diagnostické nástroje a tzv. "Performance Profiler".

8.2 Užívateľské testovanie

V ďalšej fáze testovania bola predposledná verzia programu skompilovaná a zabalená do inštalačného balíka spolu s niekoľkými ukážkovými presetmi. Tento inštalačný balík bol odoslaný jednotlivým vybraným testerom, ktorí mali za úlohu otestovanie programu z pohľadu užívateľa.

Pre účely užívateľského testovania bol vytvorený aj krátky dotazník (uvedený v prílohe A.1), ktorý obsahoval viacero sekcií, pričom každá sekcia bola venovaná iným súčastiam programu. Štruktúra dotazníka bola inšpirovaná dotazníkom z diplomovej práce T. Trkala [27].

8.2.1 Priebeh užívateľského testovania

Testovanie prebiehalo v jednej fáze, v ktorej užívatelia otestovali program, vyplnili dotazník a odoslali odpovede. Používanie programu pri testovaní pozostávalo v otvorení programu, načítaní ľubovoľného súboru, náhodného použitia jednotlivých efektov na daný súbor (alebo využitia priložených presetov) a následné uloženie výsledného súboru. Pri používaní programu mali užívatelia prirodzene možnosť využívať aj doplnkové a vizualizačné prvky programu.

Pri testovaní bolo dôležité získať najmä nasledujúce informácie:

- Vhodnosť formátu programu
- Kvalita efektov
- Výber efektov
- Využiteľnosť vlastných glitch efektov
- Intuitívnosť a prehľadnosť grafického rozhrania
- Využiteľnosť a prevedenie vizualizačných prvkov
- Praktickosť doplnkových funkcií (ovládanie prehrávania, otvorenie súboru, ...)
- Celková využiteľnosť programu v aktuálnej podobe

Pre potrebu získať uvedené informácie bol dôležitý hlavne priložený dotazník, ktorého otázky boli prispôsobené tak, aby bolo možné získať čo najužitočnejšie informácie, ale zároveň, aby daný dotazník bol aj naďalej stručný pre zrýchlenie procesu testovania. Výsledky dotazníka sa nachádzajú v prílohe A.2.

8.2.2 Výsledky a implementácia

Výsledky boli získavané postupne po jednotlivých vyplnených dotazníkoch a prešli dôkladným vyhodnotením. Spracovanie pozostávalo z kontroly odpovedí na otázky s výberom možností a následne aj kontroly textových nepovinných odpovedí, ktoré obsahovali dodatočné informácie ohľadom určitých vlastností programu.

Po kontrole odpovedí boli na základe výsledkov prehodnotené súčasti programu, ktoré dostali negatívne hodnotenie a zároveň sme zvážili návrhy, ktoré sa nachádzali v textových odpovediach.

Vo všeobecnosti boli užívatelia spokojný hlavne s vlastnými glitch efektami. Ďalšími dobre hodnotenými vlastnosťami boli napr. kvalita výstupu, voľba efektov, vizualizačné prvky, a možnosť vytvárať presety. Spokojnosť bola nižšia pri intuitivite grafického rozhrania a najhoršie bol hodnotený formát nášho programu. Formát bol hlavný dôvod, prečo by niektorí užívatelia program nevyužili v jeho aktuálnej podobe.

Ďalší krok predstavoval implementáciu / opravu / doplnenie daných súčastí programu, pričom tento krok bol z väčšej časti úspešný. Súčasti, ktoré boli doplnené, sú nasledovné:

- *Drag and Drop* funkcionalita - možnosť otvárať súbory pretiahnutím daného súboru z ľubovoľného priečinka do okna programu (šetrí čas)
- *Reset* funkcia - možnosť zmeniť stav všetkých efektov na pôvodný pomocou jedného tlačidla (šetrí čas)
- *Tooltip* funkcia - prepínateľné zobrazenie plávajúceho miniatúrneho okna pri kurzore, ktoré po prejdení nad určitým prvkom zobrazí jeho krátky popis (zvýšenie intuitivity a orientácie v programe)

Jediný návrh, ktorý nebol implementovaný do ďalšej verzie programu je zmena formátu nášho programu na VST Plugin. Zmena do iného formátu by vyžadovala redefiníciu základnej štruktúry programu a takmer všetkých tried a funkcií. Vzhľadom na vysokú časovú náročnosť tejto zmeny bol návrh zaznamenaný a pridaný medzi možné rozšírenia programu v budúcnosti (viz sekcia 9.2).

Kapitola 9

Záver

9.1 Zhrnutie

V tejto bakalárskej práci sme vytvorili hudobný softvérový program pre úpravu digitálnych zvukových stôp. Program je implementovaný ako samostatná aplikácia pre OS Windows. Implementácia prebehla v jazyku C++ vo vývojovom prostredí *Visual Studio* s využitím frameworku *JUCE*. Program obsahuje množstvo hudobných efektov. Medzi nimi sú klasické bežne používané efekty, ale aj vlastné "glitch" efekty, ktoré boli hlavným zámerom našej práce. Okrem zvukových efektov je k dispozícii aj široká škála vizualizačných prvkov. Medzi nimi sú okná s vizualizáciou časového priebehu a frekvenčného spektra signálu, a tiež frekvenčnej odozvy filtrov. Program obsahuje aj prehrávač so základným ovládaním. Zvukové stopy je možné načítavať a ukladať do výstupného súboru vo WAV alebo FLAC formáte. Okrem daných funkcií sú dostupné aj ďalšie funkcie pre zlepšenie používania, a to indikácia *clipping-u*, reset všetkých efektov, nápovede pre jednotlivé prvky a možnosť používania užívateľských presetov. Niektoré presety sú priložené v inštalácii programu, ale ďalšie je možné ľubovoľne vytvárať.

Grafické rozhranie bolo vytvorené s ohľadom na existujúce programy a preto obsahuje prvky familiárne pre bežných producentov elektronickej hudby. Program bol pri vývoji testovaný v dvoch fázach. Vlastné testovanie overilo program z pohľadu výkonu a záťaže na systém. Užívateľské testovanie formou použitia programu a vyplnenia dotazníka slúžilo pre overenie praktickosti a využiteľnosti nášho programu.

9.2 Budúca práca

Výsledný program môže byť v budúcnosti rozšírený o ďalšie prvky. Istými rozšíreniami, ktoré by prispeli k využiteľnosti a úspešnosti nášho programu sú návrhy zistené pri užívateľskom testovaní. Program je v aktuálnej forme použiteľný pre menšie množstvo producentov, a to najmä kvôli svojmu formátu. Prvou zmenou by mohlo byť prevedenie programu do formátu *pluginu* kvôli možnosti využitia v DAW softvéri. Táto ponúknutá zmena by prispela k použiteľnosti nášho programu najviac zo zistených návrhov. Druhou zmenou by bola jemná úprava grafického rozhrania, a to konkrétne zvýraznenie ovládacích prvkov. Táto zmena by prispela k zvýšeniu intuitivity u začínajúcich užívateľov nášho programu. Posledná zmena by zahŕňala vytvorenie väčšieho množstva užívateľských presetov. Vďaka ďalším presetom by užívatelia mali lepšiu predstavu o výsledkoch, ktoré naš program dokáže dosiahnuť.

Literatura

[1] O. Smith III, J. *The BiQuad Section, in Introduction to Digital Filters with Audio Applications* [online]. Center for Computer Research in Music and Acoustics (CCRMA), Stanford University, 2007 [cit. 2023-04-13]. Dostupné z: https://ccrma.stanford.edu/~jos/filters/BiQuad_Section.html.

[2] Redmon, N. *Biquad formulas, in EarLevel Engineering* [online]. Nigel Redmon, 2011 [cit. 2023-04-13]. Dostupné z: https://www.earlevel.com/main/2011/01/02/biquad-formulas/.

[3] Lutus, P. *BiQuadDesigner* [online]. Paul Lutus, 2019 [cit. 2023-04-13]. Dostupné z: https://arachnoid.com/BiQuadDesigner/index.html.

[4] O. Smith III, J. *Chorus Effect, in Physical Audio Signal Processing* [online]. Center for Computer Research in Music and Acoustics (CCRMA), Stanford University, 2010 [cit. 2023-04-13]. Dostupné z: https://ccrma.stanford.edu/~jos/pasp/Chorus_Effect.html.

[5] McAllister, M. *What Is Audio Clipping and Why Is It Important?* [online]. Produce Like A Pro, 2018 [cit. 2023-05-03]. Dostupné z: https://producelikeapro.com/blog/audio-clipping/.

[6] Glitchmachines, LLC. *Cryogen* [online]. Glitchmachines, LLC, 2015 [cit. 2023-04-27]. Dostupné z: https://glitchmachines.com/products/cryogen/.

[7] Eargle, J. *Handbook of Recording Engineering*. 4. vyd. New York : Van Nostrand Reinhold, 1996. 313–324 s. ISBN 0387284702.

[8] Glitchmachines, LLC. *Fracture* [online]. Glitchmachines, LLC, 2015 [cit. 2023-04-27]. Dostupné z: https://glitchmachines.com/products/fracture/.

[9] Tarr, E. *Full-Wave Rectification* [online]. Hack Audio, LLC, 2020 [cit. 2023-05-03]. Dostupné z: https://www.hackaudio.com/digital-signal-processing/distortion-effects/full-wave-rectification/.

[10] O. Smith III, J. *Freeverb, in Physical Audio Signal Processing* [online]. Center for Computer Research in Music and Acoustics (CCRMA), Stanford University, 2010 [cit. 2023-04-25]. Dostupné z: https://ccrma.stanford.edu/~jos/pasp/Freeverb.html.

[11] Tarr, E. *Half-Wave Rectification* [online]. Hack Audio, LLC, 2020 [cit. 2023-05-03]. Dostupné z: https://www.hackaudio.com/digital-signal-processing/distortion-effects/half-wave-rectification/.

[12] Image Line Software. *Fruity Fast Dist* [online]. 2022 [cit. 2023-05-04]. Dostupné z: https://www.image-line.com/fl-studio-learning/fl-studio-online-manual/html/plugins/Fruity%20Fast%20Dist.htm.

[13] Image Line Software. *Fruity Flangus* [online]. 2022 [cit. 2023-05-04]. Dostupné z: https://www.image-line.com/fl-studio-learning/fl-studio-online-manual/html/plugins/Fruity%20Flangus.htm.

[14] Storer, J. *JUCE Features* [online]. 2023 [cit. 2023-05-03]. Dostupné z: https://juce.com/juce/features/.

[15] Storer, J. *JUCE Tutorial: Visualise the frequencies of a signal in real time* [online]. 2020 [cit. 2023-04-16]. Dostupné z: https://docs.juce.com/master/tutorial_spectrum_analyser.html.

[16] Storer, J. *JUCE Licensing FAQ* [online]. 2017 [cit. 2023-05-03]. Dostupné z: https://juce.com/get-juce/.

[17] Storer, J. *JUCE Tutorial: Processing audio input* [online]. 2020 [cit. 2023-04-24]. Dostupné z: https://docs.juce.com/master/tutorial_playing_sound_files.html.

[18] Storer, J. *JUCE Tutorial: Draw audio waveforms* [online]. 2020 [cit. 2023-04-24]. Dostupné z: https://docs.juce.com/master/tutorial_audio_thumbnail.html.

[19] Matthe, M. *Spectral Leakage, Zero-Padding and Frequency Resolution* [online]. DSPIllustrations.com, 2018 [cit. 2023-05-05]. Dostupné z: https://dspillustrations.com/pages/posts/misc/spectral-leakage-zero-padding-and-frequency-resolution.html.

[20] Glitchmachines, LLC. *Polygon* [online]. Glitchmachines, LLC, 2020 [cit. 2023-04-27]. Dostupné z: https://glitchmachines.com/products/polygon/.

[21] O. Smith III, J. *Schroeder Reverberators, in Physical Audio Signal Processing* [online]. Center for Computer Research in Music and Acoustics (CCRMA), Stanford University, 2010 [cit. 2023-05-03]. Dostupné z: https://ccrma.stanford.edu/~jos/pasp/Schroeder_Reverberators.html.

[22] O. Smith III, J. *Soft Clipping, in Physical Audio Signal Processing* [online]. Center for Computer Research in Music and Acoustics (CCRMA), Stanford University, 2010 [cit. 2023-04-13]. Dostupné z: https://ccrma.stanford.edu/~jos/pasp/Soft_Clipping.html.

[23] Plugin Boutique Limited. *Glitch Machine Multi-Effect by Stagecraft* [online]. Plugin Boutique Limited, 2017 [cit. 2023-04-27]. Dostupné z: https://www.pluginboutique.com/product/2-Effects/53-Multi-Effect-/2349-Glitch-Machine.

[24] Thompson, D. M. *Understanding audio*: *getting the most out of your project or professional recording studio*. 1. vyd. Berklee Press, 1999 [cit. 2023-05-03]. 294–296 s. ISBN 0634009591.

[25] Thompson, D. M. *Understanding audio*: *getting the most out of your project or professional recording studio*. 1. vyd. Berklee Press, 1999. 289 s. ISBN 0634009591.

[26] Tracktion Software Corporation. *Recording Engineer* [online]. 2022 [cit. 2023-05-04]. Dostupné z: https://www.tracktion.com/products/waveform-free-expansions.

[27] Trkal, T. *Softwarový multiefekt pro postprodukci populární hudby.* Brno, CZ, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/19492/.

Příloha A

Užívateľské testovanie programu

A.1 Dotazník

Dotazník je rozdelený na 4 sekcie a každá obsahuje otázky s iným zameraním. Prvá sekcia obsahuje 2 otázky o producentskom pozadí užívateľa.

- Ako dlho sa venujete hudobnej produkcii? (Počet rokov)
- Využívate samostatné (standalone) programy pri produkcii? (pravidelne / občas / nie)

Druhá sekcia obsahuje otázky ohľadom grafického rozhrania programu. Odpovede na tieto otázky sú áno / skôr áno / skôr nie / nie.

- Je orientácia v grafickom rozhraní dostatočne ľahká a rýchla?
- Ste spokojný s umiestnením, rozmermi a dizajnom grafických prvkov?
- Sú textové popisy dostatočne čitateľné?
- Je ovládanie programu dostatočne intuitívne?

Tretia sekcia obsahuje otázky ohľadom zvukového výstupu a výkonu programu. Dostupné odpovede sú výborné / dobré / dostatočné / nedostatočné / zlé.

- Ako hodnotíte voľbu a počet efektov?
- Ako hodnotíte kvalitu zvukového výstupu efektov?
- Ako hodnotíte vizualizáciu zvukovej vlny zvukového signálu?
- Ako hodnotíte vizualizáciu frekvenčného spektra zvukového signálu?
- Ako hodnotíte ovládanie prehrávania súboru?
- Ako hodnotíte možnosť vytvárania presetov?
- Ako hodnotíte efekty Stutter, Shifter, Reverz a Extractor?

Štvrtá sekcia obsahuje dve otázky a to ohľadom formátu programu. Odpoveď je áno / nie.

- Využili by ste program v aktuálnom formáte pri hudobnej produkcii?
- Využili by ste program, ak by bol v inom formáte (VST, AAX, AU ...)?

A.2 Výsledky dotazníka

Andrej Srna
– hudobný producent, DJ, majiteľ hudobného vydavateľstva

Ako dlho sa venujete hudobnej produkcii? (Počet rokov)	4
Využívate samostatné (standalone) efekty pri produkcii?	nie
Je orientácia v grafickom rozhraní dostatočne ľahká a rýchla?	skôr áno
Ste spokojný s umiestnením, rozmermi a dizajnom grafických prvkov?	skôr áno
Sú textové popisy dostatočne čitateľné?	áno
Je ovládanie programu dostatočne intuitívne?	skôr nie
Ako hodnotíte voľbu a počet efektov?	dobré
Ako hodnotíte kvalitu zvukového výstupu efektov?	dobré
Ako hodnotíte vizualizáciu zvukovej vlny zvukového signálu?	dobré
Ako hodnotíte vizualizáciu frekvenčného spektra zvukového signálu?	dobré
Ako hodnotíte ovládanie prehrávania súboru?	dostatočné
Ako hodnotíte možnosť vytvárania presetov?	dobré
Ako hodnotíte efekty Stutter, Shifter, Reverz a Extractor?	výborné
Využili by ste program v aktuálnom formáte pri hudobnej produkcii?	nie
Využili by ste program, ak by bol v inom formáte (VST, AAX, AU, ...)?	áno

Marek Mesároš
– začínajúci hudobný producent

Ako dlho sa venujete hudobnej produkcii? (Počet rokov)	0 (3 mes.)
Využívate samostatné (standalone) efekty pri produkcii?	nie
Je orientácia v grafickom rozhraní dostatočne ľahká a rýchla?	skôr nie
Ste spokojný s umiestnením, rozmermi a dizajnom grafických prvkov?	skôr áno
Sú textové popisy dostatočne čitateľné?	áno
Je ovládanie programu dostatočne intuitívne?	skôr nie
Ako hodnotíte voľbu a počet efektov?	výborné
Ako hodnotíte kvalitu zvukového výstupu efektov?	dobré
Ako hodnotíte vizualizáciu zvukovej vlny zvukového signálu?	dobré
Ako hodnotíte vizualizáciu frekvenčného spektra zvukového signálu?	výborné
Ako hodnotíte ovládanie prehrávania súboru?	dobré
Ako hodnotíte možnosť vytvárania presetov?	dobré
Ako hodnotíte efekty Stutter, Shifter, Reverz a Extractor?	výborné
Využili by ste program v aktuálnom formáte pri hudobnej produkcii?	nie
Využili by ste program, ak by bol v inom formáte (VST, AAX, AU, ...)?	áno

Daniel Paulovič
– programátor, skúsenosti s hudobnými aplikáciami

Ako dlho sa venujete hudobnej produkcii? (Počet rokov)	0
Využívate samostatné (standalone) efekty pri produkcii?	áno
Je orientácia v grafickom rozhraní dostatočne ľahká a rýchla?	skôr áno
Ste spokojný s umiestnením, rozmermi a dizajnom grafických prvkov?	skôr áno
Sú textové popisy dostatočne čitateľné?	áno
Je ovládanie programu dostatočne intuitívne?	skôr nie
Ako hodnotíte voľbu a počet efektov?	dobré
Ako hodnotíte kvalitu zvukového výstupu efektov?	dobré
Ako hodnotíte vizualizáciu zvukovej vlny zvukového signálu?	výborné
Ako hodnotíte vizualizáciu frekvenčného spektra zvukového signálu?	dobré
Ako hodnotíte ovládanie prehrávania súboru?	dostatočné
Ako hodnotíte možnosť vytvárania presetov?	výborné
Ako hodnotíte efekty Stutter, Shifter, Reverz a Extractor?	dobré
Využili by ste program v aktuálnom formáte pri hudobnej produkcii?	áno
Využili by ste program, ak by bol v inom formáte (VST, AAX, AU, ...)?	áno

Prvá odpoveď má hodnotu 0, pretože nie je “producent” elektronickej hudby.

Příloha B

Užívateľské presety

Preset *chaoticorder.xml*

HCD	Threshold="0.5"		
Reverz	Skew="0.0"	Amount="16.0"	
Stutter	Amount="8.0"	Chorus="20.0"	Delay="99.0"
Shifter	Amount="4.0"	Tone="2.0"	

Preset *chaoticorder.xml* vytvára chaotické opakovanie s reverzovaním úsekov a zmenou frekvencie signálu. Efekty rozdeľujú signál na rovnaké úseky, čiže výsledok je chaotický a zároveň usporiadaný.

Preset *morejamming.xml*

Extractor	Intensity="35.0"	Width="5.0"
Reverz	Skew="-5.0"	Amount="24.0"

Preset *morejamming.xml* vytvára efekt vynechávania úsekov signálu a zároveň aj reverzného prehrávania niektorých úsekov.

Preset *theyarecoming.xml*

Gain	Value="-2.89"		
HCD	Threshold="0.25"		
Reverz	Skew="0.0"	Amount="16.0"	
Stutter	Amount="8.0"	Chorus="19.0"	Delay="30.0"

Preset *theyarecoming.xml* využíva hlavne Chorus efekt z efektu Stutter a Hardclip efekt pre dosiahnutie úplného "zničenia" upraveného zvuku.

Preset *drumloopvariation.xml*

Reverz	Skew="-5.0"	Amount="8.0"
Shifter	Amount="8.0"	Tone="2.0"

Preset *drumloopvariation.xml* vytvára jednoduchú pravidelnú variáciu pomocou reverzácie a zvýšenia výšky zvuku. Je vhodný pre zvukové stopy so zvukmi bicích nástrojov.

Preset *robots.xml*

Reverb	Balance="0.25"	Size="0.0"	Width="0.5"	Damp="1.0"
Stutter	Amount="4.0"	Chorus="10.0"	Delay="25.0"	

Preset *robots.xml* vytvára "robotický" efekt pomocou Stutter-a s nízkou hodnotou parametra delay. Efekt je doplnený o jemný Reverb pre zväčšenie šírky výsledného zvuku.

Preset *guitarvariation.xml*

Reverz	Skew="-5.0"	Amount="4.0"	
Stutter	Amount="4.0"	Chorus="10.0"	Delay="25.0"
Shifter	Amount="8.0"	Tone="2.0"	

Preset *guitarvariation.xml* je určený pre gitaru a iné krátke zvuky. Vytvára pravidelné variácie pomocou troch rôznych glitch efektov.

Preset *bigpaddist.xml*

Gain	Value="-3.0"			
HPF	Cutoff="200.0"	Q="1.0"		
Reverb	Balance="0.5"	Size="0.8"	Width="0.8"	Damp="0.25"
HRD	Enabled="1.0"			
Stutter	Amount="4.0"	Chorus="15.0"	Delay="99.0"	

Preset *bigpaddist.xml* využíva najmä Reverb pre vytvorenie výrazného priestorového efektu. Half-rect slúži pre výrazné skreslenie a Stutter pre pridanie variácií. Výstup je zmiernený pomocou LPF a Gain efektu. Preset je vhodný pre zvuky typu "pad".

Preset *neurobassloopvar.xml*

SCD	Threshold="0.75"	
Reverz	Skew="-5.0"	Amount="4.0"
Shifter	Amount="16.0"	Tone="1.5"

Preset *neurobassloopvar.xml* je určený pre basové slučky zložené z modulovaných basových liniek používaných v modernej elektronickej hudbe. Vytvára variácie pomocou glitch efektov a zvýrazňuje výsledný zvuk cez Softclip efekt.

Příloha C

Obsah priloženej SD karty

- `app` - výsledný .exe program a inštalačný súbor (použitý pri odosielaní programu testerom)
- `documentation/manual.pdf` - manuál programu
- `documentation/documentation.html` - odkaz na dokumentáciu (ktorá je v priečinku documentation_source)
- `JUCE` - knižnice JUCE frameworku
- `latex` - zdrojové súbory textu práce
- `M47X - GM/Builds/VisualStudio2019` - projekt pre Visual Studio 2019
- `M47X - GM/JuceLibraryCode` - inštrukcie pre JUCE (zoznam použitých knižníc)
- `M47X - GM/Presets` - užívateľské presety
- `M47X - GM/Source` - vlastné zdrojové kódy
- `misc/example_output` - niektoré ukážky upravených zvukových stôp
- `misc/test_samples` - niektoré ukážky použité pri testovaní
- `misc/video` - prezentačné video
- `pdf` - výsledný text práce v pdf formáte