

VYSOKÁ ŠKOLA POLYTECHNICKÁ JIHLAVA

Aplikovaná informatika

**Použití MATLABu při vývoji a prototypů
zvukových zásuvných modulů**

Semestrální práce

Autor práce: Aleksandr Shabelnikov

Vedoucí práce: Ing. Lucie Šaclová

Jihlava 2023

Obsah

Úvod	5
1.1 Zvukové efekty	5
1.2 Digital Audio Workstation (DAW)	5
1.3 Audio plugins	5
1.4 Audio Toolbox.....	6
1.5 Cíl	6
2 Implementace některých efektů	7
2.1 Zpracování MID-SIDE	7
2.2 Tremolo	9
2.3 Echo	11
2.4 Vibrato	14
2.5 Chorus effect	16
3 Generování kódu nebo modulu plug-in	18
3.1 validateAudioPlugin.....	18
3.2 audioTestBench	19
3.3 generateAudioPlugin	19
3.4 Praktické využití vytvořených plug-inů v DAW.....	20
Závěr	22
Seznam použité literatury	22

Úvod

1.1 Zvukové efekty

Zvukové efekty používají všichni, kdo se podílejí na vytváření hudebních signálů, a začínají u speciálních technik hry hudebníků, přechází k používání speciálních mikrofonních technik a přechází k efektovým procesorům pro syntézu, nahrávání, produkci a vysílání hudebních signálů.

Vstupní a výstupní signály jsou sledovány pomocí reproduktorů nebo sluchátek a nějakého vizuálního znázornění signálu, jako je časový signál, úroveň signálu a jeho spektrum. Podle akustických kritérií si zvukař nebo hudebník nastaví řídicí parametry pro zvukový efekt, kterého chce dosáhnout. Vstupní i výstupní signály jsou v digitálním formátu a představují analogové zvukové signály. Úprava zvukové charakteristiky vstupního signálu je hlavním cílem digitálních zvukových efektů. Nastavení řídicích parametrů často provádějí zvukaři, hudebníci (interpreti, skladatelé nebo výrobci digitálních nástrojů) nebo jednoduše posluchači hudby, ale může být také součástí jedné konkrétní úrovně v řetězci zpracování signálu digitálního zvukového efektu.

Historie audio efektů sahá až do počátku 20. století, kdy byly první zvukové efekty vytvářeny pomocí mechanických a elektrických zařízení. Například reverbace byla vytvořena pomocí ozvěny v prostoru nebo pomocí elektrických obvodů, které simulovaly ozvěnu. V 50. a 60. letech 20. století se objevily první počítačové programy pro úpravu zvuku, které umožňovaly vytvářet zvukové efekty pomocí algoritmů. Dnes jsou audio efekty běžně používány v různých oborech, jako je hudba, film a televize, a lze je najít v mnoha různých softwarových aplikacích pro úpravu zvuku.

1.2 Digital Audio Workstation (DAW)

Digitální zvuková pracovní stanice (DAW) je elektronické zařízení nebo softwarová aplikace, která slouží k nahrávání, úpravám a vytváření zvukových souborů. DAW se ovládají pomocí uživatelského rozhraní. Většina DAW umožňuje ovládání pomocí MIDI k ladění parametrů během živé editace.

V hudebním průmyslu se DAW obvykle používají k pořizování a ukládání více stop zvukových nahrávek a k mixování, ekvalizaci a přidávání zvukových efektů. DAW mají obvykle přístup ke knihovnám zvuků a používají se k vytváření elektronické hudby od základu. Komerční DAW, které se nacházejí například v nahrávacích studiích, mohou být hardwarově integrovány do počítačů.

1.3 Audio plugins

Zásuvné moduly (plugins) jsou samostatné části kódu, které lze "připojit" k DAW a rozšířit tak jejich funkčnost. Obecně zásuvné moduly spadají do kategorií zpracování zvukového signálu, analýzy nebo syntézy zvuku. Zásuvné moduly obvykle specifikují uživatelské rozhraní obsahující widgety uživatelského rozhraní, které však může být rozhraním DAW maskováno. Typické zásuvné moduly zahrnují ekvalizaci, řízení dynamického rozsahu, dozvuk, zpoždění a virtuální nástroje.

Pro zpracování streamovaných zvukových dat zavolá DAW zásuvný modul, předá snímek vstupních zvukových dat a přijme zpět snímek zpracovaných výstupních zvukových dat. Když se změní parametr zásuvného modulu (například když přesunete ovládací prvek v uživatelském rozhraní zásuvného modulu), DAW oznámí zásuvnému modulu novou hodnotu parametru. Zásuvné moduly mají obvykle vlastní uživatelské rozhraní, ale DAW poskytují také obecné uživatelské rozhraní pro všechny zásuvné moduly.

1.4 Audio Toolbox

Audio Toolbox™ poskytuje nástroje pro zpracování zvuku, analýzu řeči a akustická měření. Obsahuje algoritmy pro zpracování zvukových signálů, jako je ekvalizace a časové roztažení, odhad metrik akustického signálu, jako je hlasitost a ostrost, a extrakci zvukových vlastností, jako je MFCC a výška tónu. Poskytuje také pokročilé modely strojového učení, včetně i-vektorů, a předtrénované sítě hlubokého učení, včetně VGGish a CREPE. Aplikace Toolbox podporují živé testování algoritmů, měření impulsní odezvy a označování signálů. Toolbox poskytuje streamovací rozhraní pro ASIO™, CoreAudio a další zvukové karty; zařízení MIDI a nástroje pro generování a hostování zásuvných modulů VST a Audio Units.

Pomocí nástroje Audio Toolbox můžete importovat, označovat a rozšiřovat sady zvukových dat a také získávat funkce pro trénování modelů strojového učení a hlubokého učení. Poskytnuté předtrénované modely lze použít na zvukové nahrávky pro sémantickou analýzu na vysoké úrovni.

Můžete vytvářet prototypy algoritmů zpracování zvuku v reálném čase nebo provádět vlastní akustická měření pomocí streamování zvuku s nízkou latencí do zvukových karet a ze zvukových karet. Svůj algoritmus můžete ověřit tak, že jej proměníte ve zvukový zásuvný modul, který spustíte v externích hostitelských aplikacích, jako jsou například digitální zvukové pracovní stanice. Hostování zásuvných modulů umožňuje používat externí zvukové zásuvné moduly jako běžné objekty MATLAB®.

Audio Toolbox™ podporuje generování kódu do nejrozšířenějšího formátu zásuvných modulů VST (Virtual Studio Technology) společnosti Steinberg. Audio Toolbox také umožňuje spouštět a testovat externě vytvořené pluginy VST a VST3 přímo v MATLAB®.

1.5 Cíl

V této práci se zaměřujeme na možnosti softwaru MATLAB® pro práci s digitálními zvukovými signály. Budeme studovat některé algoritmy pro vytváření různých zvukových efektů, včetně hudebních signálů, s ohledem na různé vlastnosti hudebního materiálu, jako je tempo, rozměr a další. Tyto algoritmy a skripty také převedeme do objektově orientovaných tříd v prostředí MATLAB®, abychom mohli později vytvářet zásuvné moduly pomocí nástroje Audio Toolbox.

2 Implementace některých efektů

2.1 Zpracování MID-SIDE

Mid-side (M/S) je technika záznamu a mixování zvuku, která se používá k upravení stereo šířky zvuku. Při této technice se zvuk rozdělí na dvě složky: "mid" (střední) složku, která obsahuje zvuk ze středu stereo panorámy, a "side" (boční) složku, která obsahuje zvuk ze stran stereo panorámy. Tyto dvě složky lze poté upravovat nezávisle na sobě a následně sloučit zpět do jedné stereo stopy.

Mid-side technika se obvykle používá k upravení stereo šířky zvuku nebo k úpravě frekvenčního spektra zvuku. Například lze snížit boční složku, aby se zvuk zúžil a zněl více jako mono zvuk, nebo naopak zvýšit boční složku, aby se zvuk rozšířil a zněl více stereo.

Pokud chcete převést stereo signál na mid-side formát, můžete použít následující vzorce:

$$\text{Mid složka} = (\text{levý kanál} + \text{pravý kanál}) / 2$$

$$\text{Side složka} = (\text{levý kanál} - \text{pravý kanál}) / 2$$

Pokud chcete z mid-side formátu zpět převést na stereo formát, můžete použít následující vzorce:

$$\text{Levý kanál} = \text{mid složka} + \text{side složka}$$

$$\text{Pravý kanál} = \text{mid složka} - \text{side složka}$$

V obou případech jsou mid a side složky zvuku představovány jako samostatné audio stopy nebo proměnné.

Uvažujme následující kód v MATLAB®:

```

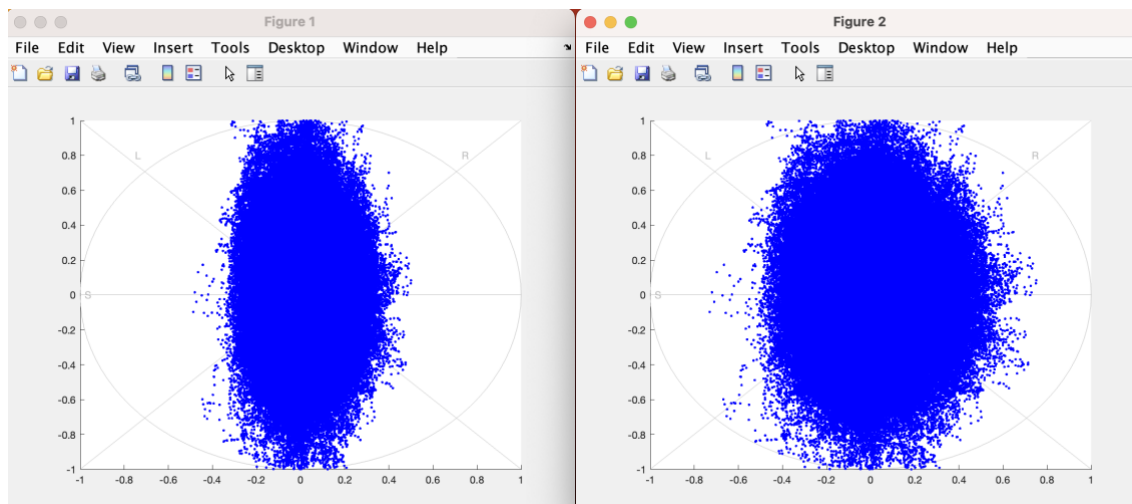
1  clear; clc; close all;
2  [x, Fs] = audioread('ptv-mix.aif');
3
4  % mid-side conversion
5  mid = (x(:,1) + x(:,2))/2; % mid = (left + right)/2
6  side = (x(:,1) - x(:,2))/2; % side = (left - right)/2
7
8  % side gain
9  side_gain = 1.5; % gain of the side channel
10
11 % process side
12 side = side * side_gain; % apply gain (m)
13
14 % convert back to stereo
15 newleft = mid + side;
16 newright = mid - side;
17
18 y = [newleft, newright];
19
20 % old and new
21 oldandnew = [x; y];
22 % play audio
23 soundsc(oldandnew, Fs);
24 % plot original and processed audio
25 goniometer(x, 1);
26 goniometer(y, 2);

```

Obr. 1: mid-side processing (stereowidth.m)

Tento kód slouží k převodu stereo signálu na mid-side formát, úpravě boční složky zvuku a následnému převodu zpět na stereo formát. Kód nejprve načte audio soubor do proměnné "x" a vytvoří mid-side složky pomocí vzorců, které jsem uvedl předem. Poté se boční složka zvuku upraví pomocí zvoleného zesílení a nakonec se mid a side složky sloučí zpět do stereo formátu pomocí dalších vzorců. Kód také zobrazí starý a nový zvuk v grafu a přehrává nově upravený zvuk.

Zkontrolujeme výsledek algoritmu pomocí nástroje Goniometr.



Obr. 2: Goniometr

Goniometr vykreslí bod pro každý časový vzorek ve stereofonním signálu. Amplituda vstupního signálu v časovém vzorku pro levý kanál je L a pro pravý kanál je R. Poloha každého bodu se určí výpočtem úhlu θ mezi levým a pravým kanálem. Konvenčně se do měření úhlu zahrne rotační posun o 45° , takže stereostřed má hodnotu 90° a stereopole je symetrické kolem svislé osy. Kromě toho se pro každý bod vypočítá kombinovaná velikost neboli poloměr jako vzdálenost od počátku. Nakonec se polární souřadnice (r, θ) převedou na kartézské souřadnice (x, y) pro vykreslení. Na pravém obrázku č. 2 (zpracovaný signál) je zřetelně vidět rozšíření stereo panoramatu (body jsou ve více krajních polohách).

$$\theta[n] = \tan^{-1} \left(\frac{L[n]}{R[n]} \right) + \frac{\pi}{4}$$

$$r[n] = \sqrt{L[n]^2 + R[n]^2}$$

$$x[n] = r[n] \cdot \cos(\theta[n])$$

$$y[n] = r[n] \cdot \sin(\theta[n])$$

Je důležité si uvědomit, že ne vždy je možné kontrolovat výsledek zvukových efektů pomocí jakýchkoli nástrojů, někdy vám spektrum signálu nebo jeho časová doména neřeknou nic, co by potvrdilo, že algoritmus pracuje správně. V tomto případě zbývá jediná kontrola — sluch.

2.1.1 Překlad kódu do objektově orientovaného stylu

Chcete-li vytvořit zásuvný modul, musíte tento kód přeložit do objektově orientovaného stylu (podrobnosti viz <https://www.mathworks.com/help/audio/gs/convert-matlab-script-to-an-audio-plugin.html>).


```

3  classdef stereoWidthPlugin < audioPlugin % inherits from audioPlugin class properties
4
5  properties (Access = public) % public properties
6      Width = 1.5;
7  end
8
9  properties (Constant) % interface
10     PluginInterface = audioPluginInterface(...
11         audioPluginParameter('Width', 'Mapping',{'pow',2, 0, 4}))
12 end
13
14 methods
15     function out = process(plugin, in) % process method
16         % Define processing here
17         mid = (in(:,1) + in(:,2))/2;
18         side = (in(:,1) - in(:,2))/2;
19         side = side * plugin.Width;
20         out = [mid + side, mid - side];
21     end
22
23     function reset(plugin) % reset method
24         plugin.Width = 1.5;
25     end
26
27     function set.Width(plugin, width) % set method
28         plugin.Width = width;
29     end
30 end
31
32 end
33

```

Obr. 3: stereoWidthPlugin.m

Tento kód definuje třídu "stereoWidthPlugin", která dědí od třídy "audioPlugin". Třída má jednu veřejnou vlastnost nazvanou "Width" a konstantní vlastnost "PluginInterface", která obsahuje objekt třídy "audioPluginInterface". Třída také obsahuje tři metody: "process", "reset" a "set".

Základní metoda process (nezbytná metoda pro každý zvukový plug-in), která obsahuje algoritmus rozkladu M/S, mění hodnotu boční složky a převádí zpět na stereofonní signál.

2.2 Tremolo

Tremolo je audio efekt, který mění hlasitost zvuku v pravidelných intervalech. Tento efekt se často používá v hudební produkci k vytvoření dojmu pulsace nebo k dosažení jiného zvukového efektu.

Princip tremola je založen na opakovaném zvýšení a snížení hlasitosti zvuku pomocí křivky modulačního signálu. Modulační frekvence a hloubka modulace se obvykle mohou nastavit, takže lze přizpůsobit vzhled a charakter tremola podle potřeby.

Tremolo se může použít na jednotlivé zvukové stopy nebo na celý mix, a může být také kombinováno s jinými audio efekty, jako je například reverbace nebo delay, pro vytvoření složitějších zvukových efektů.

Amplitudová modulace (AM) je způsob, jakým se mění hlasitost zvuku v závislosti na křivce modulačního signálu. Princip amplitudové modulace je založen na opakovaném zvýšení a snížení hlasitosti zvuku pomocí křivky modulačního signálu. Modulační frekvence a hloubka modulace se obvykle mohou nastavit, takže lze přizpůsobit vzhled a charakter amplitudové modulace podle potřeby.

V následujících příkladech nebudu uvádět celý kód. Všechny příklady, včetně těch předchozích, jsou na GitHubu na adrese <https://github.com/mrmidi/VSPJ-PTV/SEMESTRAL>. Přesto se zde budu věnovat některým důležitým částem.

```

16 properties (Access = private)
17     sampleRate = 44100; % Sample rate (default)
18     currentPhase = 0; % Current phase of the LFO
19     angleChange = 0.1 * (1 / 44100) * 2 * pi; % Angle change per sample
20 end

```

Obr. 4: tremoloPluginV2.m

Tento kód definuje tři vlastnosti v rámci třídy. Vlastnost "sampleRate" představuje vzorkovací frekvenci, tedy počet vzorků zvuku, které jsou zaznamenány za sekundu. Vlastnost "currentPhase" představuje aktuální fázi LFO (nízkofrekvenční oscilátor), který se používá k generování modulačního signálu. Vlastnost "angleChange" představuje změnu úhlu v každém vzorku, která se používá při výpočtu fáze LFO.

```

35 for n = 1:numSamples
36     % Calculate the LFO value
37     if strcmp(plugin.Waveform, 'sine')
38         lfo = (plugin.Depth / 2) * sin(plugin.currentPhase) + (1 - plugin.Depth / 2); % Sine wave
39     else

```

Obr. 5: tremoloPluginV2.m

Tento kód vypočítá hodnotu LFO (nízkofrekvenční oscilátor) pomocí sinusové funkce a aktuální fáze LFO uložené v plugin.currentPhase. Hodnota LFO je vypočítána jako součin hloubky modulace (plugin.Depth) a sinusové funkce aktuální fáze, který je následně sečten s jedním minus hloubka modulace dělená dvěma. Výsledná hodnota LFO tedy představuje sinusovou křivku modulačního signálu s hloubkou modulace definovanou v plugin.Depth.

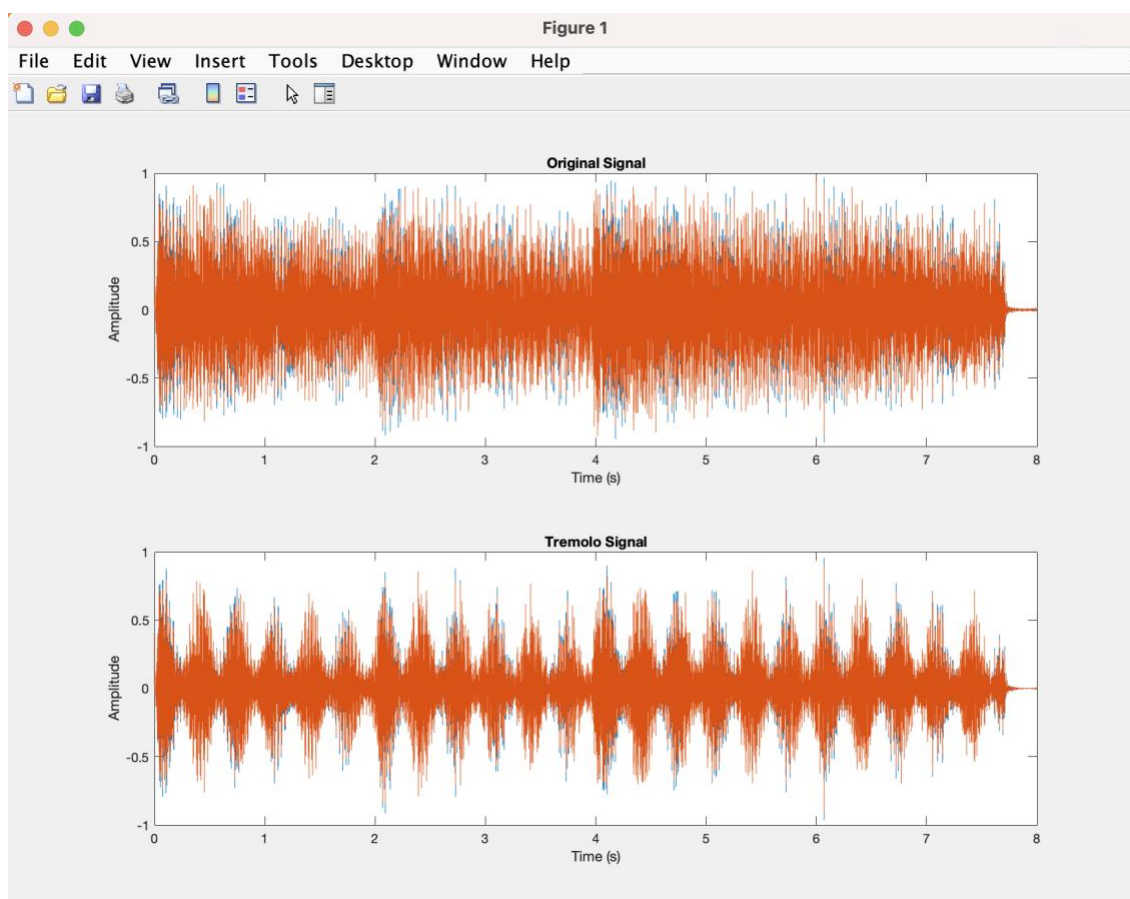
```

43     % Apply the LFO to the input
44     out(n,:) = in(n,:) * lfo;
45
46     % Update the phase
47     plugin.currentPhase = plugin.currentPhase + plugin.angleChange; % Update the phase of the LFO
48     if plugin.currentPhase > 2 * pi % Wrap phase if necessary
49         plugin.currentPhase = plugin.currentPhase - 2 * pi; % Wrap phase
50 end

```

Tento kód se používá k aplikaci modulace na zvukový vstup. V prvním řádku se hodnota LFO použije k modulaci zvukového vstupu "in" a výsledek se uloží do výstupního pole "out".

V následujících řádcích se pak aktualizuje fáze LFO pomocí "angleChange" a kontroluje, zda nedosáhla hodnoty $2 \cdot \pi$. Pokud ano, je fáze "wrappována" zpět na začátek. Toto se opakuje pro každý vzorek zvuku v poli "in", takže výsledné pole "out" obsahuje zvukový vstup modulovaný LFO. Jinými slovy, k výpočtu modulace se zde místo času používají vlastnosti trigonometrické kružnice.



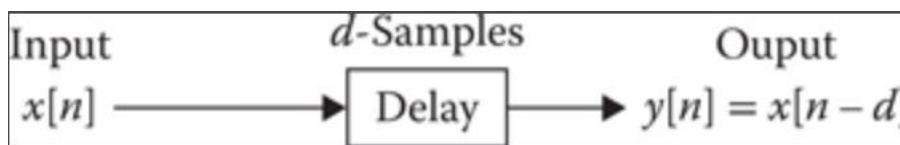
Obr. 6: Výsledek efektu: spodní obrázek jasně ukazuje změnu amplitudy v závislosti na dané modulaci.

2.3 Echo

Všechny dříve popsané systémy jsou založeny na zpracování podle prvků (element-wise). Ekvivalentní kategorizací je popsat tyto systémy jako systémy bez paměti. Jinými slovy, výstup v libovolném okamžiku ze systému závisí pouze na vstupu do systému ve stejném okamžiku. Tyto systémy nevyžadují ke zpracování signálu paměť předchozích časových vzorků.

Ve zvuku se používá mnoho dalších systémů zpracování signálu, kde výstup v libovolném čase může záviset na vstupním signálu v předchozím čase. Tyto systémy vyžadují paměť pro uložení vstupních vzorků pro pozdější použití. Jedná se o systémy s pamětí.

Systémy s pamětí mají schopnost dočasně uložit signál a časově ho odložit. Blok zpožděného zpracování přijímá vzorek vstupního signálu při čísle vzorku n a ukládá tento vzorek do paměti na určitý počet vzorků d . Při čísle vzorku $n + d$ je výstupem bloku zpožděného zpracování vstupní vzorek $x[n]$. Když tedy blok zpožděného zpracování přijme vstupní vzorek $x[n]$, výstupem bloku je současně vzorek $x[n - d]$, neboli vzorek uložený v paměti po dobu d vzorků. Blokové schéma základního systému se zpožděním je znázorněno na obrázku č. 7.



Obr. 7: Blokové schéma základního systému se zpožděním

2.3.1 Převod tempa na vzorky

Jednou z technik, kterou používají zvukoví inženýři u efektů ozvěny, je synchronizace doby zpoždění s tempem skladby. Opakování signálu tedy probíhá v rytmu s vystoupením. Synchronizovaná doba zpoždění se vypočítá převodem tempa v úderech za minutu na vzorky pomocí následujících vztahů:

$$X \frac{\text{beats}}{\text{minute}} \cdot \frac{1 \text{ minute}}{60 \text{ sec}} = \frac{\text{beats}}{\text{sec}}$$

$$\frac{1}{\frac{\text{sec}}{\text{beat}}} = \frac{\text{sec}}{\text{beat}}$$

$$\frac{\text{sec}}{\text{beat}} \cdot \frac{F_s \text{ samples}}{\text{sec}} = Y \frac{\text{samples}}{\text{beat}}$$

```

8      [x, Fs] = audioread('rhodes.aif');
9
10     bpm = 120; % BPM of the beat
11
12     bps = bpm/60; % Beats per second
13
14     spb = 1/bps; % Seconds per beat
15
16     |
17     % whole = 4, half = 2, quarter = 1, eighth = 0.5, sixteenth = 0.25
18     noteLength = 1/3; % Length of each note in beats
19
20     % calculate delay times in seconds
21     delayTime1 = spb * noteLength;
22
23     % convert to samples
24     d = fix(delayTime1 * Fs); % delay in samples, rounded down
25
26     N = length(x); % length of input signal
27     y = zeros(N, 2); % initialize output signal
28

```

Obr. 8: tremolo.m

Tento kód slouží k výpočtu délky delay efektu v sekundách a v počtu vzorků pro zvukový soubor "rhodes.aif". K výpočtu se používá počet úderů za minutu (BPM), délka jedné noty v úderech a vzorkovací frekvence zvukového souboru "Fs".

Nejprve se v proměnné "bpm" uloží počet úderů za minutu a v proměnné "bps" se vypočítá počet úderů za sekundu. Poté se v proměnné "spb" vypočítá počet sekund na jeden úder. Délka jedné noty v úderech je uložena v proměnné "noteLength" a je vyjádřena jako zlomek z celé noty (například osminová nota je noteLength = 0.5).

Poté se v proměnné "delayTime1" vypočítá délka delay efektu v sekundách jako produkt počtu sekund na jeden úder a délky jedné noty v úderech. Poté se v proměnné "d" vypočítá počet vzorků, které odpovídají délce delay efektu, a hodnota je zaokrouhlená dolů.

2.3.2 Percepční časová fúze

Časová fúze je pro posluchače percepčním jevem. Základním předpokladem tohoto jevu je, že posluchači percepčně slučují opakování signálu s krátkým časovým zpožděním do jednoho sluchového obrazu.

Důležitým faktorem při časové fúzi je posluchačův práh ozvěny. Pokud je časová prodleva mezi opakováním menší než práh ozvěny, je vnímán jeden zvuk namísto samostatných, zpožděných opakování. Pokud je časové zpoždění mezi opakováním delší než práh ozvěny, je vnímána vícenásobná ozvěna.

Časové zpoždění prahu ozvěny posluchače je ~30 ms. Přesné časové zpoždění závisí na několika věcech, včetně obálky signálu attack, decay, sustain, release (ADSR), relativní amplitudy opakování a vnímané stereofonní separace opakování.

2.3.3 Feedback echo

Feedback echo je efekt, který se používá ke generování opakování zvuku po určitém časovém intervalu. Tento efekt se obvykle používá v aplikacích pro zpracování zvuku nebo v hudebních nástrojích, jako jsou například efektové pedály pro elektrické kytary.

Zpětná vazba je jedním z klíčových prvků feedback echo efektu. Zpětná vazba se používá k opakování zvuku po určitém časovém intervalu, což vytváří dojem, že zvuk je "zpožděn" v čase. Čím vyšší je úroveň zpětné vazby, tím více opakování zvuku se vytvoří.

Výstup systému, $y[n]$, lze zapsat jako funkci vstupního signálu, $x[n]$, a minulých vzorků ze zpožďovacích bloků. Tento způsob reprezentace systému se nazývá diferenční rovnice. Popis pomocí diferenční rovnice poskytuje cestu k implementaci systému v počítačovém kódu. $x[d]$ je buffer pro základní signál.

$$y[n] = x[n] + g \cdot x[d]$$

```

42      % create a buffer to hold the delayed signal
43      buffer = zeros(d, 2);
44
45      g = 0.5; % feedback gain
46
47      % loop through input signal
48
49      for n = 1:N
50          y(n, :) = x(n, :) + g * buffer(end, :);
51          buffer = [y(n, :); buffer(1: end-1, :)];
52      end

```

Obr. 9: tremoloPluginV2.m

Tento kód slouží k vytvoření delay efektu pro zvukový vstup "x" s použitím zpětné vazby "g". V prvním řádku se vytvoří pole "buffer", které bude sloužit jako "paměť" pro ukládání vzorků zvuku pro opakované použití. V proměnné "g" je uložen koeficient zpětné vazby, který určuje, jak silně bude zpětná vazba ovlivňovat výsledný zvuk.

V cyklu "for" se pro každý vzorek zvuku v poli "x" vypočítá výstupní vzorek "y" jako součet zvukového vstupu "x" a zpětné vazby "g" na poslední vzorek v poli "buffer". Poté se v poli "buffer" posune obsah o jedno místo dopředu a na začátek se přidá nový vzorek "y". Tento proces se opakuje pro každý vzorek zvuku v poli "x", takže výsledné pole "y" obsahuje zvukový vstup s delay efektem.

Z tohoto obrázku nemůžeme posoudit správnost algoritmu: vidíme, že se signál změnil, ale nemůžeme přesně říci jak. V následujících příkladech nebudu uvádět vizualizaci sigly v časové oblasti.

2.4 Vibrato

Následující uvažované efekty lze považovat za kombinaci efektů zpoždění a amplitudové modulace. Sériově modulované zpoždění se při použití jako zvukový efekt nazývá vibrato. Ačkoli je tento efekt založen na časovém zpoždění, obvykle se nepoužívá k vytvoření slyšitelné ozvěny. Místo toho vibrato zavádí mírné změny výšky signálu při zvyšování nebo snižování zpoždění. Když se zpoždění zvětšuje (tj. prodlužuje), je to, jako by se signál zpomaloval. Proto se perioda signálu prodlužuje a frekvence klesá. Když se zpoždění zmenšuje (tj. zkracuje), je to, jako by se signál zrychloval. Proto se perioda signálu zkracuje a frekvence se zvyšuje.

2.4.1 Doppler Effect

Souvisejícím konceptem akustických vln je Dopplerův jev. Když se zdroj zvuku pohybuje směrem k pozorovateli, frekvence signálu se zvyšuje, protože se vlnová délka stlačuje. Když se zdroj zvuku od pozorovatele vzdaluje, frekvence signálu klesá, protože se vlnová délka rozšiřuje.

2.4.2 Parametry vibráta

V praxi prochází efekt vibráta obvykle vzorcem zvyšující se a snižující se frekvence. Změny efektu jsou řízeny pomocí LFO. Amplituda a frekvence LFO jsou parametry pro modulaci doby zpoždění. Tyto parametry se běžně označují jako hloubka (amplituda) a rychlost (frekvence). Konceptně se parametr hloubky používá k nastavení toho, v jak širokém rozsahu se mění výška tónu, a parametr rychlosti se používá k nastavení toho, jak rychle se mění výška tónu.

2.4.3 Circular Buffer

Circular buffer, také známý jako kruhový buffer nebo cyklický buffer, je speciální typ paměti, který se často používá v aplikacích pro zpracování zvuku nebo v hudebních nástrojích. Je to paměťová struktura, která se chová jako kruh, což znamená, že po dosažení konce paměti se adresy začnou opět počítat od začátku.

Circular buffer se často používá k ukládání vzorků zvuku pro opakované použití, jako je například v případě delay efektu nebo reverbu. Tento typ paměti má několik výhod oproti běžným lineárním pamětem, jako je například menší potřeba přesouvat data a rychlejší přístup k datům.

Circular buffer se obvykle implementuje jako pole nebo vektor, kde se používá ukazatel na aktuální pozici v poli. Po vyčerpání kapacity bufferu se ukazatel automaticky přesune na začátek

bufferu a začne se znovu počítat. Tím se zajistí, že nové vzorky zvuku se přidávají na začátek bufferu a staré vzorky se postupně "vysunou" z konce bufferu.

```

1 function [out, buffer] = vibratoEffect(in, buffer, Fs, n, depth, rate)
2
3     t = (n-1)/Fs;
4     lfo = (depth / 2) * sin(2 * pi * rate * t) + depth;
5
6     len = length(buffer);
7
8     indexC = mod(n-1, len) + 1; % Current index
9
10    fracDelay = mod(n-lfo-1, len) + 1; % Fractional delay
11    intDelay = floor(fracDelay); % Integer delay
12    frac = fracDelay - intDelay; % Fractional part
13
14    nextSamp = mod(intDelay, len) + 1; % Next sample
15
16    out = (1 - frac) * buffer(intDelay, :) + frac * buffer(nextSamp, :);
17
18    buffer(indexC, :) = in; % Update buffer
19
20 end

```

Obr. 10: tremoloEffect.m

Tento kód slouží k aplikaci vibrato efektu na zvukový vstup "in" s použitím paměti "buffer", vzorkovací frekvence "Fs", aktuálního vzorku "n", hloubky vibrato "depth" a frekvence vibrato "rate". Vibrato efekt se tvoří modulací délky delay efektu pomocí tzv. low-frequency oscillator (LFO).

Nejprve se vypočítá aktuální čas "t" v sekundách jako poměr aktuálního vzorku "n" a vzorkovací frekvence "Fs". Poté se pomocí LFO vypočítá hodnota "lfo", která se použije k modulaci délky delay efektu.

Dále se zjistí délka bufferu "len" a aktuální index "indexC" pro ukládání nových vzorků do bufferu. Poté se pomocí LFO vypočítá celočíselná část "intDelay" a zbytek "frac" pro výpočet frakčního delay efektu. V tomto konkrétním případě se interpolace používá k výpočtu výstupního vzorku "out" pro daný vzorek zvuku "in" s použitím frakčního delay efektu.

V kódu se interpolace počítá následovně:

$$\text{out} = (1 - \text{frac}) * \text{buffer}(\text{intDelay}, :) + \text{frac} * \text{buffer}(\text{nextSamp}, :);$$

Zde:

- out je výstupní vzorek, který se vypočítá pomocí interpolace
- frac je frakční část delay efektu, která se vypočítá jako rozdíl mezi celočíselnou částí "intDelay" a frakční částí "fracDelay"
- buffer(intDelay, :) je vzorek z bufferu pro celočíselnou část delay efektu
- buffer(nextSamp, :) je vzorek z bufferu pro frakční část delay efektu

Výsledný vzorek "out" se tedy vypočítá jako interpolace mezi vzorky pro celočíselnou a frakční část delay efektu pomocí koeficientu "frac". Tento koeficient určuje, jaký podíl má každý vzorek na výsledném vzorku "out".

Pro zjištění správné pozice vzorku v kruhové vyrovnávací paměti kód používá modulo (vydělí pozici délkou vyrovnávací paměti).

Nemá smysl zobrazovat výsledek algoritmu vizuálně, doporučuji spustit skript tremolo.m v MATLAB®.

2.5 Chorus effect

Chorus efekt je zvukový efekt, který se používá k vytvoření dojmu sboru nebo hlasů v unisonu. Chorus efekt se často používá v hudbě pro vytvoření plnějšího a bohatšího zvuku kytary nebo klávesových nástrojů.

Chorus efekt se vytvoří tak, že se zvukový signál duplikuje a jedna kopie se moduluje v čase nebo frekvenci. Výsledný zvuk se pak složí se zvukovým vstupem, což vytvoří dojem, že zvuk pochází ze sboru nebo více hlasů. Je to, jako by byl part dvakrát nebo třikrát nahraný (tj. dvakrát nebo třikrát) a vrstvený, aby se zvuk zdánlivě "zahustil".

Chorus efekt se může použít pro širokou škálu hudebních žánrů, od popu a rocku po elektronickou hudbu.

Efekt chorus úzce souvisí s efektem vibrato. Chorus vzniká paralelním smícháním nezpracovaného suchého vstupního signálu s efektem vibrato. Doba zpoždění efektu se obvykle pohybuje v rozmezí 10-50 ms. Tento rozsah se blíží prahu ozvěny posluchače, takže efekt nevytváří slyšitelnou ozvěnu v důsledku delšího časového zpoždění.

2.5.1 Parametry funkce Chorus

Parametry efektu chorus jsou podobné jako u efektu vibrato. Rate a depth upravují amplitudu, respektive frekvenci LFO, který se používá k modulaci doby zpoždění. Kromě toho lze parametr pre-delay použít jako amplitudový offset tak, aby se doba zpoždění pohybovala v rozmezí 10-50 ms.

```

60 for n = 1:numSamples
61
62
63     % calculate delay time
64
65     lfoMS = p.Depth * sin(p.currentPhase) + p.Predelay; % in ms
66     fracDelay = mod(p.DelayLineIndex - lfoMS, len) + 1; % Fractional delay
67     intDelay = floor(fracDelay); % Integer delay
68     frac = fracDelay - intDelay; % Fractional part of delay
69
70

```

Obr. 11: chorusPluginV4.m

Tento kód slouží k výpočtu delay efektu s použitím modulace hloubky "Depth" a frekvence "currentPhase" pomocí LFO. Delay efekt se používá k zpoždění zvukového signálu a jeho opakovanému zpětnému použití v čase.

Nejprve se vypočítá hodnota "lfoMS" pro modulaci hloubky delay efektu pomocí LFO. Tato hodnota je v milisekundách a je vypočítána jako součin hloubky "Depth" a sínusové hodnoty LFO "currentPhase" s následným přidáním předzpoždění "Predelay".

Poté se pomocí funkce "mod" vypočítá frakční část delay efektu "fracDelay" a z ní se odvodí celočíselná část "intDelay" a frakční část "frac". Tyto hodnoty se použijí pro interpolaci vzorků v poli "buffer" při výpočtu výsledného zvuku s delay efektem.

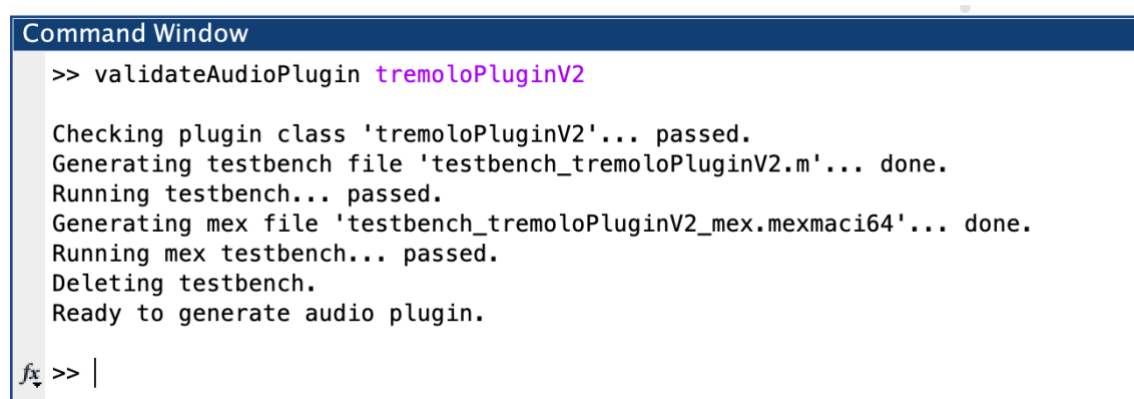
Zde je vidět, že algoritmus je velmi podobný vibratu, hlavním rozdílem je přítomnost parametru Predelay. Doporučuji poslechnout si rozdíl spuštěním skriptu chorus.m.

3 Generování kódu nebo modulu plug-in

Všechny diskutované efekty lze převést do formátu plug-in nebo zdrojového kódu pro následnou práci v různých IDE. Základními nástroji pro práci s pluginy v MATLAB® jsou příkazy `validateAudioPlugin`, `generateAudioPlugin`, `audioTestBench`. Prozkoumejme je jednu po druhé.

3.1 `validateAudioPlugin`

Vygeneruje a spustí proceduru Test Bench, která otestuje vaši třídu zvukového zásuvného modulu. Při vývoji zásuvných modulů je nutné tento nástroj spustit, protože další práce v DAW. Například při použití špatného kódu ve vyšších jazycích může dojít k zavěšení programu nebo celého operačního systému. Kontrola nezaručuje správnost práce algoritmu, ale ochrání před některými kritickými chybami.



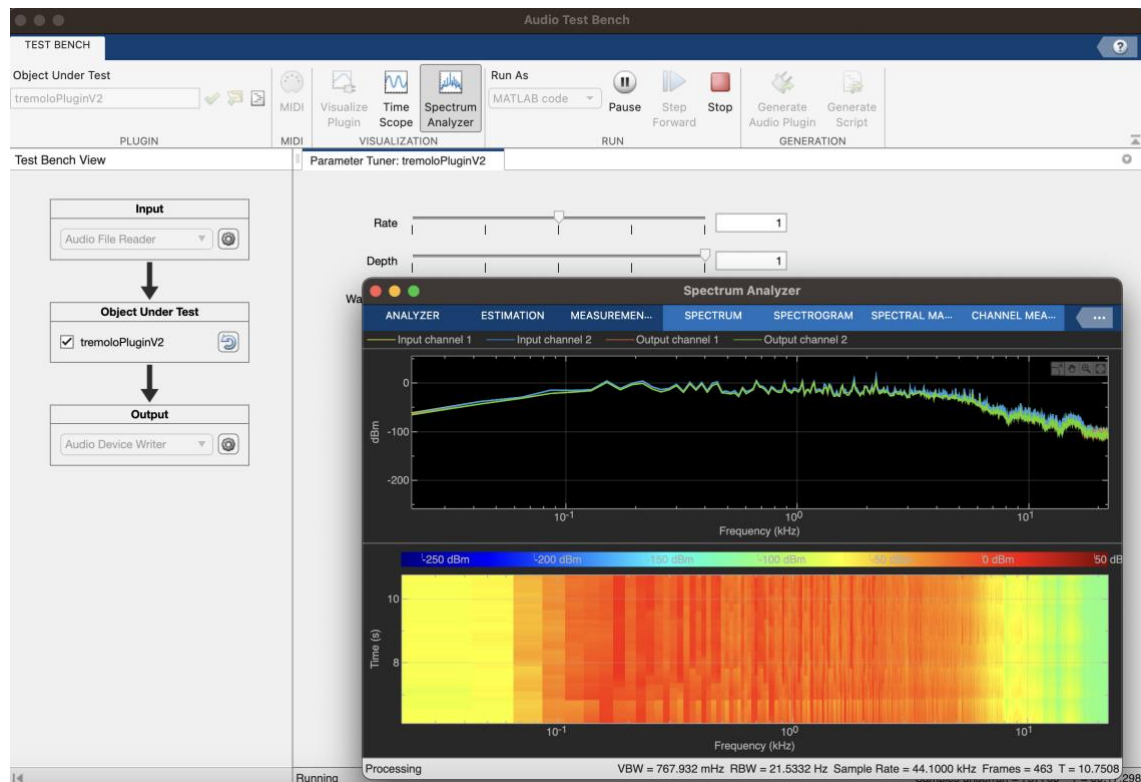
```
Command Window
>> validateAudioPlugin tremoloPluginV2

Checking plugin class 'tremoloPluginV2'... passed.
Generating testbench file 'testbench_tremoloPluginV2.m'... done.
Running testbench... passed.
Generating mex file 'testbench_tremoloPluginV2_mex.mexmaci64'... done.
Running mex testbench... passed.
Deleting testbench.
Ready to generate audio plugin.

fx >> |
```

Obr. 12: Příklad provedení příkazu `validateAudioPlugin`. V tomto případě kontrola nezjistila žádné kritické chyby.

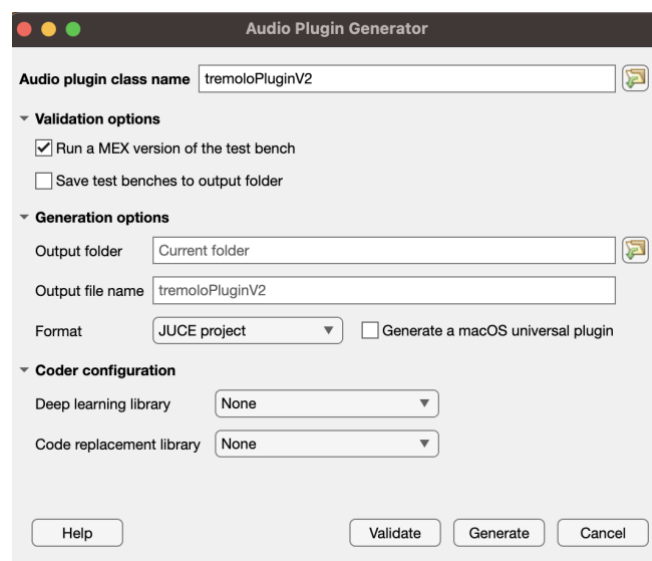
3.2 audioTestBench



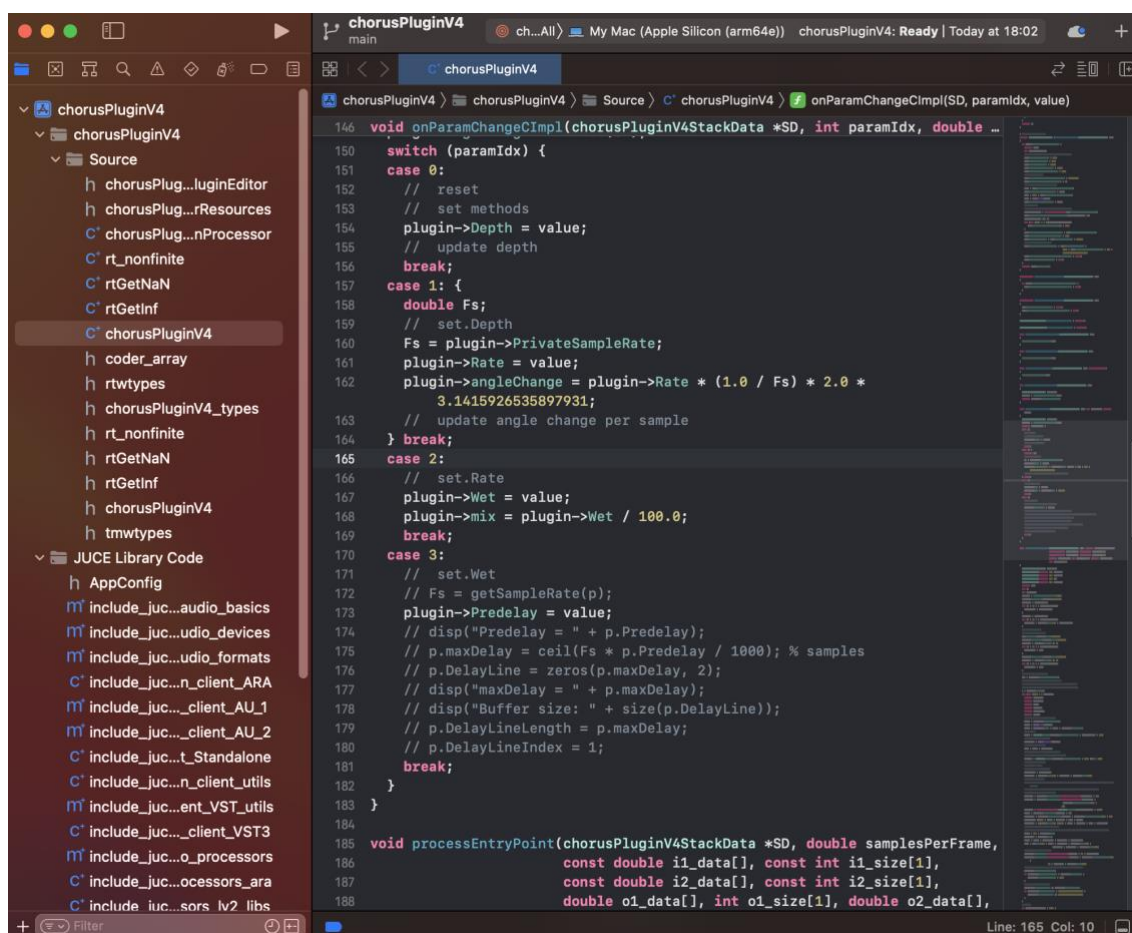
Obr. 13: nástroj Audio Test Bench

3.3 generateAudioPlugin

Nástroj pro generování zdrojového kódu zvukového plug-inu nebo C++. Při generování zdrojového kódu bude kód generován pomocí populárního real-time audio frameworku JUCE (<https://www.juce.com>). Počínaje verzí MATLAB® 2022b má nástroj grafické uživatelské rozhraní, viz obr. 14. Na obrázku 15 je vygenerovaný zdrojový kód zásuvného modulu otevřen ve vývojovém prostředí XCode. Dokonce i komentáře byly přeneseny.



Obr. 14: Audio Plugin Generator



Obr. 16: XCode

3.4 Praktické využití vytvořených plug-inů v DAW

Vygenerované moduly plug-in v systému Mac OS musí být umístěny v adresáři /Library/Audio/Plug-ins/Components. Na obrázku 17 používám efekty, které jsem napsal v MATLAB® v profesionálním hudebním a zvukovém systému Logic Pro.



Obr. 17: Logic Pro

Závěr

Tento článek se zabývá základními zvukovými efekty. Z hlediska požadavků je teoretická část značně zjednodušená, ale některé algoritmy jsou ukázány v praxi. Na závěr je třeba poznamenat, že mnoho efektů je kombinací jednoduchých komponent: například Vibrato a Chorus jsou kombinací zpoždění a modulace. Mnoho zajímavých algoritmů, které se používají při zpracování digitálního zvukového signálu, jako je Reverb, ekvalizace, hřebenové filtry, FIR a IIR filtry atd., zde nebylo zahrnuto.

Lze říci, že MATLAB® je výjimečný nástroj pro práci s hudbou a dalšími zvukovými signály. Neuvažovali jsme o takových nástrojích, jako je DSP Toolbox, který obsahuje všechny výše uvedené efekty a umožňuje je používat pomocí jediného řádku kódu (viz `dsp.VariableFractionalDelay`, `dsp.FrequencyDomainFIRFilter`, `dsp.AsyncBuffer` atd.). Seznam všech signálových a zvukových nástrojů by vydal na samostatný článek. Generování kódu do jazyka C++ dělá z MATLAB® standard pro prototypování, analýzu a vývoj zvukových efektů a syntezátorů. V současné době neznám žádný podobný nástroj: například v Pythonu je snadné pracovat se zvukem pomocí některých knihoven, ale možnosti MATLABu® jsou tomu velmi vzdálené.

Všem zájemcům o tento materiál doporučuji nahlédnout do odkazů uvedených na konci této práce.

Seznam použité literatury

DeVane, Charlie, and Gabriele Bunkheila. "Automatically Generating VST Plugins from MATLAB Code." Audio Engineering Society Convention 140. Audio Engineering Society, 2016.

BENNETT, Christopher. Digital Audio Theory: A Practical Guide. Focal Press, 2020. ISBN 978-1-119-99130-4.

TARR, Eric. Hack Audio: An Introduction to Computer Programming and Digital Signal Processing in MATLAB® (Audio Engineering Society Presents). Routledge, 2018. ISBN 978-1-351-01846-3.

MÜLLER, Meinard. *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications*. Springer-Verlag New York, 2015. ISBN 978-3-319-21945-5. Dostupné z: doi:10.1007/978-3-319-21945-5

