VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

EXPERIMENTÁLNÍ SOFTWAROVÝ HUDEBNÍ NÁSTROJ ŘÍZENÝ BARVOU SVĚTLA
EXPERIMENTAL SOFTWARE MUSICAL INSTRUMENT CONTROLLED BY LIGHT COLOR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Matej Liska**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. MgA. Mgr. Dan Dlouhý, Ph.D.**
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Audio inženýrství**
specializace Zvuková produkce a nahrávání
Ústav telekomunikací

Student: Matej Liska ***ID:*** 203739
Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Experimentální softwarový hudební nástroj řízený barvou světla

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je kompletně naprogramovat experimentální elektroakustický softwarový hudební nástroj. Jeho experimentálnost spočívá v netradičním způsobu řízení parametrů zvukové syntézy pomocí analýzy barev, obsažených ve vložených obrázcích. Student realizuje funkční aplikaci, vycházející z návrhu, provedeného v semestrální práci.

DOPORUČENÁ LITERATURA:

[1] PUCKETTE, M. Theory and Techniques of Electronic Music, 2006. 337 s. online: http://msp.ucsd.edu/techniques.htm

[2] FORRÓ D. Svět MIDI. Grada, Praha, 1997. 375s. ISBN 80-7169-412-6.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 8.6.2020

Vedoucí práce: doc. Ing. MgA. Mgr. Dan Dlouhý, Ph.D.

doc. Ing. Jiří Schimmel, Ph.D.
předseda rady studijního programu

ABSTRAKT

Bakalárska práca zahŕňa teoretické podklady pre realizáciu experimentálneho zvukového softvérového nástroja riadeného farbou svetla. Približuje poznatky z problematiky zvukového a svetlého vlnenia a spracovania farieb. Mimo iné popisuje vytvorenie digitálnych zvukových signálov použitím aditívnej syntézy, ich možné signálové spracovanie a oboznamuje s technikou sonifikácie. Rieši programovacie možnosti vytvárania, spracovania, prehrávania a ukladania digitálnych zvukových signálov. Zvukový zdroj je tvorený v jazyku Java na programovacej platforme JavaFX.

KĽÚČOVÉ SLOVÁ

aditivnívna syntéza, signálové spracovanie, sonifikácia, Java, JavaFX

ABSTRACT

Bachelor thesis contains the theory for realization of experimental software instrument controlled by light color. It brings knowledge of sound and light waves and color processing. Besides others, it describes process of generating digital signals using additive synthesis, methods of signal processing and sonification techniques. It solves programming possibilities of creating, processing, playing and saving digital audio signals. The audio source is written in Java language on the JavaFX programming platform.

KEYWORDS

additive synthesis, signal processing, sonification, Java, JavaFX

LISKA, Matej. *Experimentální softwarový hudební nástroj řízený barvou světla*. Brno, 2020, 60 s. Bakalárska práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedúci práce: doc. Ing. MgA. Mgr. Dan Dlouhý, Ph.D.

VYHLÁSENIE

Vyhlasujem, že svoju bakalársku prácu na tému „Experimentální softwarový hudební nástroj řízený barvou světla“ som vypracoval samostatne pod vedením vedúceho bakalárskej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej bakalárskej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto bakalárskej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno 7.6.2020 .....

podpis autora

POĎAKOVANIE

Ďakujem vedúcemu bakalárskej práce doc. Ing. MgA. Mgr. Danovi Dlouhému, Ph.D., za pedagogickú a odbornú pomoc, trpezlivosť, návrhy a konzultáciu pri tvorbe práce.

Obsah

Zoznam obrázkov

Zoznam výpisov

Úvod

Cieľom bakalárskej práce je realizovať experimentálny softvérový hudobný nástroj riadený farbou svetla využívajúci prvky aditívnej syntézy obohatenej o signálové spracovanie s multimediálnym grafickým užívateľským rozhraním určeným pre počítačovú platformu. Práca popisuje teoretické poznatky potrebné k realizácii nástroja umožňujúceho užívateľovi vkladať obrázkové súbory, z ktorých je vytvorená zvuková ukážka, v závislosti na farebných vlastnostiach pixelov a počte pixelov vloženého súboru.

Vygenerovaný digitálny zvukový záznam je možné upravovať jednotlivými typmi signálových procesov a úpravy medzi sebou kombinovať. Nástroj ponúka možnosť rozdelenia vloženého obrázkového média na viaceré časti, vytvárajúce samostatné zvukové ukážky s taktiež individuálnou možnosťou spracovania. Možnosť prehrávania vytvorenej zvukovej ukážky je sprístupnená po importovaní obrázka a taktiež po jednotlivých zmenách nastavenia nástroja. Výsledný zvuk je umožnené ukladať pre ďalšie možnosti spracovania.

V prvej časti práca rieši základné teoretické poznatky z oblasti vlnení, jeho typov a vlastností. Zaoberá sa problematikou zvukového a svetelného vlnenia. Poukazuje na možné riešenie prepojenia medzi frekvenciami počuteľného a viditeľného spektra. Skúma problematiky vzťahov medzi svetlom a farbami. Poukazuje na možnosti vytvárania farieb a mapovania do akronym.

Druhá časť práce je zameraná na problematiku z oblasti vytvárania a spracovania digitálneho zvukového média. Táto časť práce približuje informácie ohľadom tvorby zvuku využitím syntéz a podrobnejšie popisuje realizáciu aditívnej syntézy. Spôsobom spracovania digitálneho zvuku procesom filtrácie zobrazuje teoretické vlastnosti a kompletný návrh pre praktickú aplikáciu. Taktiež sa zaoberá problematikami techniky sonifikácie, rieši jej teoretické poznatky a dôvod použitia v tejto práci. Poukazuje na vlastnosti a problematiky dátových formátov vrátane špecifikácie zvukového formátu použitého pri tvorbe zvukovej ukážky nástrojom.

Návrhová časť predstavuje blokovú schému nástroja vrátane podrobného popisu funkčnosti a komunikácie jednotlivých častí blokov nástroja.

Kapitola 4 - Realizácia práce je zameraná na využitie teoretických poznatkov a celého návrhu pre vytvorenie softvérového nástroja vrátane podrobného popisu jednotlivých krokov a výpisov kódu. Zobrazuje grafický návrh užívateľského prostredia, približuje dôvody využitia programovacieho jazyka Java a popisuje spôsoby algoritmizácie jednotlivých častí nástroja vrátane zdrojových kódov.

Kapitola 5 - Zhodnotenie poukazuje na vlastnosti vytvoreného nástroja zo strany autora práce. Zameriava sa na popísanie výsledných zvukových ukážok, zvukových možností, predkladá zmeny medzi návrhom a realizáciou a možné využitie nástroja.

Táto práca kombinuje vlastnosti tvorby umelého zvuku, spracovania signálov a programovania. Skúma preklad jednotlivých farieb do zvukových dát, ich následné spracovanie vrátane vplyvu na zvukové spektrum a vnímanie ľudským sluchom.

Nástroj v porovnaní s inými voľnými softvérovými aplikáciami rieši rozloženie farebného spektra a zameriava sa na preklad do zvukovej oblasti pomocou teoretického poznatku o svetelnom spektre, ktoré sa nachádza frekvenčne položené o 40 oktáv vyššie ako zvukové spektrum. Rieši problematiku farieb, ktoré sa nenachádzajú v tomto spektre. Nezaoberá sa tvarmi vloženého obrázkového média, ale jeho farebnými vlastnosťami. Využíva teoretické poznatky z oblasti syntéz, filtrácie a sonifikácie, ktoré aplikuje v programovej podobe pri tvorbe nástroja. Umožňuje užívateľovi spoznať, použiť a sluchovo analyzovať jednotlivé signálové procesy. Poskytuje možnosti riešenia individuálnych častí obrázkového média nezávisle na iných, vrátane zapamätania nastavených údajov jednotlivých segmentov, čím je možné dosiahnuť separátny hudobný vnem každej časti obrázka individuálne, s možnou spätnou reflexiou.

Vytvorená zvuková ukážka je vo forme samplu, čo značí zvukovú časť zachytávajúcu farbu nástroja (podrobnejšie popísané v kapitole 5). Možnosťou uloženia práca vytvára a zapisuje na počítačové úložisko zvukový formát bezstratovej kompresie.

Nástroj bol navrhnutý a realizovaný na základe experimentálneho pokusu o vytvorenie novej možnosti prepojenia medzi obrazom a zvukom. Pre jeho širšie využitie boli pridané jednotlivé možnosti spracovania obrázka aj zvuku, výsledný zvukový formát je možno použiť pre ďalšie spracovanie alebo vytváranie hudobných skladieb alebo aplikáciu iného výsledku v závislosti od kreativity užívateľa.

Práca sa zaoberá využitím programovacieho jazyka Java bez externých knižníc k tvorbe a spracovaniu obrazu a zvuku. Grafická časť pre možnosti ovládania je vytvorená pomocou platformy JavaFX.

1 Vlnenie zvuku, svetla a definícia farieb

Nasledujúca kapitola práce je zameraná na vysvetlenie teórie problematiky vlnenia, budú predstavené dva rozličné typy vlnení. V prvej časti bude pre úplnosť a logické uvedenie do kontextu vysvetlená problematika mechanického vlnenia a podrobne opísaná teória zvukových vĺn. V ďalšej časti bude predstavená a vysvetlená teória z oblasti elektromagnetických vĺn a viditeľného spektra. Taktiež sa zameriame na vysvetlenie problematiky viditeľného spektra a riešenie prevodu svetla do farieb. Kapitola oboznamuje s teoretickými poznatkami ohľadom definície farieb, ich aspektami a mapovaním do akronym.

1.1 Vlnenie a jeho typy

Vlnenie je jedným z najrozšírenejším fyzikálnym javom. Stretávame sa s ním v podobe zvuku, svetla alebo rozhlasového či televízneho vysielania. [1]

Rozlišujeme dva typy šírenia vlnenia. Ak sú okamžité výchylky kolmé na smer šírenia, hovoríme o priečnom vlnení. Ak sa jedná o rovnobežné výchylky so smerom šírenia, hovoríme o pozdĺžnom vlnení. [2]

1.1.1 Zvukové vlnenie

Zvukom nazývame každé mechanické vlnenie, ktoré je schopné vyvolať v ľudskom uchu sluchový vnem. [1]

Zvukové vlny sa šíria prevažne pozdĺžnym vlnením, v pevných látkach aj ako priečne vlnenie. Rýchlosť šírenia zvuku závisí od hustoty média, v ktorom je vlnenie prenášané. Menením energie vloženej do vlny ovplyvňujeme jej intenzitu, na rýchlosť šírenia zmena energie vplyv nemá. [3]

Pri pozdĺžnom typu vlnenia vzniká v prostredí prídavný premenný tlak vyvolaný procesom zhusťovania alebo zriedovania častíc prostredia, nazývame ho akustický tlak. Napriek tomu, že nie všetky vlnenia sú nezávislé od tvaru vlny, zvuk túto vlastnosť má. Rýchlosť pre basové či výškové frekvencie zvuku je vo vzduchu zaokrúhlene 340 metrov za sekundu. Zvuk je teda vibrácia fyzického média, ako je napríklad vzduch, voda alebo hélium. Môžeme tiež povedať, že zvukové vlny pozostávajú zo zhusťovania a zriedovania prostredia. [3]

Procesom zmeny hustoty prostredia je ovplyvňovaný pohyb ušných bubienkov, ktoré pohybom dovnútra alebo von premieňajú toto mechanické vlnenie na elektrické signály spracovávané v ľudskom mozgu. Disciplínu, skúmajúcu problematiku zvuku a zvukových vlnení sa nazývame akustika. [1]

1.1.2 Svetelné vlnenie

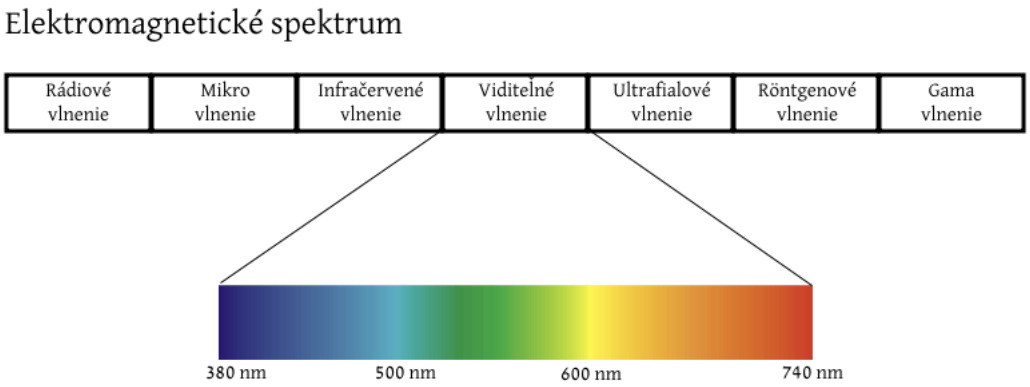
Svetelné vlnenie je časťou elektromagnetického spektra. Elektromagnetickým vlnením označujeme každý dej v premennom elektromagnetickom poli, ktorého zmeny sa šíria priestorom. Elektromagnetické vlnenie je vlnenie priečneho typu, ktoré charakterizujú dve vektorové zložky. [1]

Vektor **B** magnetickej indukcie, je magnetická zložka vlnenia a vektor **E**, ktorý označuje zložku elektrickej indukcie vlnenia. Tieto dva vektory sú na seba kolmé. [4]

Svetelné vlnenie ako časť elektromagnetického žiarenia nadobúda vlnové dĺžky približne od 380 nm do 780 nm. [5]

Vedeckí pracovníci sa doteraz presne nezhodli na konkrétnom vlnovom rozpätí viditeľného spektra. Napriek tomu, že meriame energiu obmedzenú len na rozsah týkajúci sa viditeľného svetla, nenachádzame priamu súvislosť medzi množstvom energie a vnímanou intenzitou. [6]

V rôznych odborných materiáloch možno nachádzať len približné hodnoty a rozpätia vlnových dĺžok svetelného vlnenia.



Obr. 1.1: Znázornenie viditeľného vlnenia v elektromagnetickom spektre.

[vytvorené autorom práce]

1.2 Problematika svetla a farieb

Svetlo a farba sú pojmami, ktoré je možno vnímať ľudským okom, komentovať ich, diskutovať alebo odkazovať na ne. Diskusie, ktoré tieto problematiky riešia, často končia nezhodou alebo nedorozumením, ktoré je založené na skutočnosti, že oba pojmy majú viacero protichodných významov.

„Osvetľovací technik vie presne, čo svetlo je a zároveň architekt presne vie, čo svetlo je, ale hovoria o rôznych veciach.“ (cit. Liljenfors; ARNKIL, Harald: *Colour and Light*, str. 45, 2010.) Koncepčný zmätok spôsobuje problémy, pokiaľ ide o kvantifikáciu množstva svetla, diskusiu o kvalite svetla alebo o určenie presnej farby a jej charakteristík. [6]

Pojem vnímané svetlo označuje pozorované svetlo, ktoré možno opísať vlastnosťami, ako je napríklad úroveň, distribúcia, tiene, odrazy, oslnenie a hlavne farba svetla. Všetky tieto koncepcie, ktoré boli špecificky preskúmané a prezentované, označujú všetky aspekty svetla ako vizuálny zážitok, aspekty, ktoré nie je možné identifikovať alebo kvantifikovať iným spôsobom, ako dôsledným vizuálnym pozorovaním. [6]

1.2.1 Možnosti definície farby

Napriek ťažkostiam pri definovaní farby je možné definovať rôzne atribúty farby oveľa presnejšie. Tieto jednotlivé farebné atribúty sú pojmami, ktoré majú pri modelovaní farebného vzhľadu mimoriadny význam. [7]

Odtieň je atribút farby na základe vizuálneho pocitu, podľa ktorého sa oblasť javí podobná jednej z vnímaných farieb: červená, žltá, zelená a modrá alebo kombinácia dvoch z nich. Tieto farby sa nazývajú základné farby.

Farebnosť je atribút určujúci pomer, či je farebná oblasť viac alebo menej chromatická.

Chromatickosť je farebnosť oblasti posudzovaná ako podiel jasu podobne osvetlenej oblasti, ktorá sa javí ako biela alebo vysoko priepustná.

Jasnosť je atribút farby, podľa ktorého oblasť pravdepodobne vyžaruje viac či menej svetla.

Svetlosť je jas oblasti posudzovanej vzhľadom na jas podobne osvetlenej oblasti, ktorá sa javí ako biela alebo vysoko priepustná.

Vyfarbenie je atribút posudzujúci farebnosť farby vzhľadom na jej jasnosť.

Aj keď je farba zvyčajne považovaná za trojrozmernú zloženinu a farebné zhody je možné špecifikovať iba tromi číslami, ukázalo sa, že pri niektorých prípadoch tri rozmery nestačia na úplné určenie farebného vzhľadu. [7]

1.2.2 Aplikácie farieb

V priemyselných aplikáciách sa slovo farba často týka vzorca a špecifická farba sa identifikuje pomocou kódov, ako sú kódy CMYK pre tlač alebo percentuálne hodnoty RGB pre obrazovky. To znamená, že jedna a tá istá vzorcová farba sa môže líšiť vo vizuálnom vzhľade a spektrálnom rozložení v závislosti od presnosti výroby, kalibrácie technického vybavenia, kombinácie s inými materiálmi, v ktorej sa pozoruje. [6]

Definícia farieb prostredníctvom vzorcov je nevyhnutná pri reprodukcii alebo aplikácii farieb vo viacerých odvetviach použitia. Mnoho farebných grafov a výberov farebných vzoriek je pevne zaužívaných a definovaných.

1.2.3 Frekvencie farieb

Ako bolo v práci vyššie uvedené, problematika svetla a farieb je stále vyvíjajúcou sa a spresňujúcou disciplínou. Jedným z hlavných poznatkov týkajúcich sa tejto problematiky je aj fakt, že nie všetky farby, ktoré možno ľudským okom vnímať, sa nachádzajú konkrétne v elektromagnetickom viditeľnom spektre.

V roku 1676 Isaac Newton urobil experiment, pri ktorom slnečné svetlo nechal prechádzať cez sklenený trojuholníkový hranol, ktorý spôsoboval lom svetla do spektra farieb. Toto spektrum ale neobsahovalo všetky odtiene farieb. [8]

Farby ako napríklad odtiene sivej alebo odtiene ružovej či fialovej nie je možné presne frekvenčne definovať, pretože sú vytvorené zložením farieb z viditeľného spektra. Nemajú teda vlastnú frekvenčnú hodnotu.

1.2.4 Psychofyzikálny aspekt farieb

Psychofyzikálna metodológia pre skúmanie farieb je nazývaná kolorimetria. Jej primárnym cieľom je identifikovať a kvantifikovať vizuálne rozdiely medzi farebnými stimulmi, aby sa zabezpečila stabilita výroby a aby sa stanovili úrovne tolerovanej odchýlky. [6]

Kolorimetria bola vyvinutá z rovnakých základných predpokladov pozorovateľov, ktorí boli požiadaní, aby upravili zmesi troch monochromatických svetelných stimulov tak, aby sa zhodovali s farbou jedného monochromatického stimulu. Z týchto informácií sa vypočítalo niekoľko modelov, v ktorých je každý farebný stimul charakterizovaný tromi fyzikálnymi premennými (hodnoty tristimulu), zvyčajne dominantnou vlnovou dĺžkou, jasom a spektrálnou čistotou. [9]

Tieto funkcie sú znázornené ako diagramy nazývané „farebné priestory“, ktoré je možné použiť na určenie súradníc farieb, ako je diagram CIELAB a stupnice farebných rozdielov. [6]

1.2.5 Mapovanie farieb do akronym

V prvom rade je nutné poznamenať, že v ľudskom mozgu neexistujú žiadne kanály ako napríklad RGB či HSV, na ktorých by bolo možné založiť farebné mechanizmy počítačov a nástrojov na meranie farby. Všetky geometrické znázornenia farieb sú preto kompromisom medzi jasnosťou a presnosťou. Vo farebných priestoroch sa musí urobiť kompromis medzi dvomi charakteristikami: symetriou a krokmi rovnakých farebných rozdielov. Tento problém viedol k novým matematicko-geometrickým znázorneniam farebných premenných s akronymami, ako sú HSV, HSB, HSL, LCh, CIE Yxy, CIELAB, CIELUV a CIECAM. Dôvodom veľkého počtu modelov nie je zložitosť problematiky svetla, ale spočíva v nájdení modelu, dostatočného na pravdivú simuláciu ľudského farebného videnia. [6]

2 Skúmanie tvorby a digitálneho spracovania zvukových signálov

V tejto časti práce budú predstavené problematiky vytvárania signálov a ich spracovania použitím syntéz a filtrov. Vysvetlené sú pojmy z oblasti syntéz, typy syntéz, príklady použitia a podrobnejšie je priblížená problematika aditívnej syntézy, ktorá je realizovaná v tejto práci. Kapitola oboznámi s vysvetlením filtrácie zvukových vzoriek, návrhom jednotlivých typov filtrov a ich realizáciu v použití digitálnej filtrácie.

Ďalej bude v stručnosti vysvetlená technika sonifikácie, jej teoretický obsah, využitie a použitie v tejto práci.

2.1 Syntéza

Pojem syntéza možno chápať ako proces spájania jednotlivých zložiek spoločných vlastností dovedna pre vytvorenie nového celku. Formy syntéz môžu mať konštruktívne alebo aj deštruktívne následky na konečný výsledok. Pojem syntéza je najčastejšie užívaný v dvoch základných kontextoch: pri vytváraní chemických zlúčenín a pri produkcii elektronickej hudby. [10]

2.2 Hudobná syntéza

Proces produkovania hudby generovaním elektrických alebo mechanických vĺn, alebo aj len následné elektrické či mechanické procesy, v ktorých využívame už existujúce zvuky, nazývame hudobná syntéza. Proces hudobnej syntézy môže byť rozličný, a teda aj samotný výstup sa môže jednotlivo odlišovať. Podľa jednotlivých metód je možnosť produkovať jednoduché až komplexné zvuky odlišných farieb, s bohatým či ochudobneným spektrom. Hlavným prvkom vzniku hudobnej syntézy je proces zaoberajúci sa matematickými, fyzikálnymi a aj biologickými vlastnosťami umenia aplikovaného vo vede. [10]

Pod pojmom syntetizér chápeme elektronický nástroj využívajúci svoje vlastnosti k produkovaniu hudby, s viacerými možnosťami nastavenia pre jeho ovládanie a kvalitu zvuku. Syntetizované zvuky je možno rozdeliť do dvoch kategórií, ako sú imitatívne a syntetické zvuky. Nie vždy je rozdelenie jednoznačné, zvuky môžu nadobudnúť charakter oboch kategórií. Medzi imitatívne zvuky radíme tie, ktorých priebehy zvukových vĺn sa snažia napodobiť skutočné hudobné nástroje. [10]

Do kategórie syntetických zvukov radíme vytvorené zvukové ukážky, nepodobajúce sa reálnym hudobným nástrojom, a tiež šumy, ruchy a podobne. [10]

2.3 Harmonické a neharmonické zložky zvuku

Zvuky z reálneho sveta nie sú zvyčajne deterministické, teda sa neskladajú len z fundamentu a jeho celočíselných násobkov harmonických zložiek. Zloženie týchto zvukov obsahuje frekvencie, ktoré nemožno vyjadriť jednoduchými číselnými hodnotami. Obsahuje napríklad čísla racionálne alebo aj iracionálne. [10]

Harmonické zložky, ktoré nie sú celočíselnými násobkami fundamentálnej frekvencie, nazývame neharmonickými frekvenciami.

Typy neharmonických frekvencií:

- inharmonické spektrum,
- šumy,
- ruchy.

Šum nemá harmonickú štruktúru, napriek tomu môže byť prezentovaný len v niektorých pásmach spektra. Podľa toho, kde v spektre sa šum nachádza, ho následne pomenúvame farebne (napr. biely, ružový, hnedý, atď.). Môžeme teda povedať, že rôzny šum je zložený z veľkého množstva frekvencií, ktoré sa nachádzajú v danom spektrálnom pásme, aj keď ich amplitúda a fáza je čisto náhodná. [10]

Inharmonické spektrum býva časťou harmonickej rady pri spektre reálneho hudobného nástroja. Inharmonické spektrum nájdeme prevažne pri aplikácii AM alebo FM syntézy. [10]

2.4 Typy syntéz

Existuje mnoho rôznych typov syntézy, ktoré možno rozlíšiť jadrom procesu, reprodukcie alebo manipulácie pri tvorbe zvuku. [11]

Následne sú v práci spomenuté najviac používané typy syntéz a je uvedené ich základné rozdelenie.

2.4.1 Aditívna syntéza

Základným prvkom aditívnej syntézy je spájanie jednotlivých spektrálnych zložiek s rozdielnymi parametrami vlastností, ako je frekvencia či amplitúda dovedna. Výsledným priebehom je spojené široké spektrum, obsahujúce všetky vstupné signály. Riešenie aditívnej syntézy je možné realizovať analógovou alebo digitálnou formou. [10]

Využitie Fourierovej transformácie v oblasti aditívnej syntéz

Každý opakujúci sa signál, teda signál s periódou, je možno zreprodukovať. Na reprodukciu periodického signálu sa používa spájanie jednoduchých priebehov, taktiež periodických signálov opísateľných ich amplitúdou a frekvenciou. Matematický proces, pri ktorom dostávame číselné hodnoty frekvencií a amplitúdy potrebné k zreprodukovaniu signálu, sa nazýva Fourierova transformácia (FT). Takýmto spôsobom sa vytvárajú zložené priebehy signálov, ako je napríklad píla, trojuholník alebo štvorcový signál. [10]

Aditívna syntéza teda nie je systém obsahujúci všetky prvky k tvoreniu plne reálneho zvuku, keďže vychádza len z možných číselných hodnôt, zároveň ale nie je pravidlom sčítavať len celočíselné násobky fundamentu, a výsledný zvuk obsahujúci inak matematicky zložené spektrum harmonických zložiek môže vo výsledku vytvoriť nový, jedinečný zvuk.

Matematické vyjadrenie

Proces riešenia aditívnej syntézy pomocou matematického vyjadrenia sa realizuje pomocou prvkov, ktoré nadobúdajú rozličné hodnoty. Prenos tak dostane výstupný syntetický prvok, ktorý je tvorený zo vstupných parametrov. [12]

Rovnicu využívajúcu prvky aditívnej syntézy možno vyjadriť vzťahom (2.1). [12]

$$\begin{aligned} y(t) &= \sum_{k=1}^{K} A_k(t) \sin(\omega_k(t)), \\ \omega_k(t) &= 2\pi f_k(t) + \varphi_k, \quad t > 0 \end{aligned} \tag{2.1}$$

Jednotlivé členy majú tento význam:

y – výstupná hodnota matematickej rovnice aditívnej syntézy,
t – časové číslo vzorky,
k – poradové číslo sínusoidy v zvuku,
K – konečné číslo počtu sínusov,
A_k – hodnota amplitúdy k-tej vzorky,
ω_k – uhlová frekvencia signálu k-tej vzorky,
f_k – frekvencia k-tej vzorky,
φ_k – fázový posun k-tej vzorky.

2.4.2 Subtraktívna syntéza

Subtraktívna syntéza využíva pre svoj proces spektrálne bohatý signál, ktorý následne filtruje, respektíve odkresáva, jeho spektrálne zložky, pre tento proces využíva jednotlivé filtre. Spektrálne širokým signálom teda rozumieme signál s veľkým množstvom harmonických zložiek, ktorý je matematicky formovaný pre vytvorenie jednotlivých finálnych priebehov, napríklad signály s priebehom píla, štvorec, trojuholník a pod. [10]

2.4.3 Wavetable syntéza

Z principiálnej myšlienky subtraktívnej syntézy vychádza funkčnosť vytvárania viac sofistikovaného zvuku pomocou wavetable syntézy, ktorá ukladá viacero cyklov vlnových foriem do tabuliek, ktoré sú následne dynamicky pridávané v reálnom čase. V digitálnej rovine sa pridáva možnosť pracovať so zvukmi v komplexných rovinách, v ktorých vznikajú krátke zvuky priradené do cyklov. [10]

2.4.4 FM a AM syntéza

Syntézy typu AM alebo FM vytvárajú jednotlivé modulácie vstupného signálu, ktorý ak sa jedná o FM syntézu, frekvenčne modulujú a ak sa jedná o AM syntézu signál modulujú amplitúdovo. Pri zvyšovaní frekvencie modulácie vznikajú nové inharmonicity. Pri frekvenčnej modulácii, kedy použijeme viacero modulačných signálov, tiež môžu nastať javy ako je otočenie fázy, zrkadlenie alebo odčítavanie jednotlivých vĺn, tieto javy sú podstatou využitia AM alebo FM syntézy. [10]

2.4.5 Sampling

Sampling je metóda založená na wavetable, kedy jednotlivé zložky umiestnené v takzvanej tabuľke, ktoré boli predtým navzorkované z pôvodného signálu, sú prehrávané opakovane pre vytvorenie základu kontinuality výsledného zvuku. Nevýhodou tohoto typu syntézy je hlavne potreba veľkého úložiska pre jednotlivé vzorky signálu. [10]

2.4.6 Granulárna syntéza

Granulárna syntéza je špecifickým typom syntézy využívajúci krátke časové časti zvuku, ktoré sú prehrávané v rýchlych intervaloch, na hranici tónu a impulzu. Tieto krátke zvuky nazývame granule alebo zrná, mávajú časovú veľkosť okolo 1 až 100 milisekúnd. Hlavným konceptom celej funkčnosti granulárnej syntézy je, že každý tón možno rozložiť na množstvo malých segmentov, meniacich sa v čase. [10]

2.4.7 Fyzikálne modelovanie

Fyzikálne modelovanie je digitálna metóda, ktorej podstatou je pokus emulovať akustický zvuk k vytvoreniu čo najpodobnejšieho zvuku ako reálny hudobný nástroj. Tento typ syntézy používa filtrovanie a obálky len vo veľmi malom množstve. [10]

Metóda spočíva v matematickom znázornení nástroja, ktorý sa má reprodukovať, s cieľom napodobniť fyzikálne správanie zvukových vĺn zo súboru premenných veličín (alebo presnejšie z ich matematického vyjadrenia), ako je veľkosť, tvar, stavebné materiály, hracia technika (vibrácie struny, perkusívny zvuk atď.) a ďalšie aspekty. Je to syntéza zvuku, ktorá pomocou súboru rovníc a algoritmov emuluje fyzické správanie zdroja zvuku. Takto je možné vytvárať precízne simulácie hudobných nástrojov (presnejšie ako vytvorené inými metódami syntézy), zvukovo reálne ako hudobný nástroj alebo experimentálne (pretože je umožnené meniť parametre a modifikovať tvar, veľkosť atď.). Prekvapujúcim aspektom tejto technológie syntézy je, že bola veľmi úspešná nielen pri emulácii akustických nástrojov, ale aj pri opätovnom virtuálnom vytvorení klasického analógového syntetizéra využívajúceho subtraktívnu syntézu. [11]

2.5 Filtrácia

Kapitola 2.5 popisuje problematiku digitálnej filtrácie (kapitola 2.5.1), použitej pri realizácii softvérového nástroja. Taktiež sú popísané typy filtrácie (kapitola 2.5.2), a konkrétnejšie IIR filtrácia, ktorá bola aplikovaná v našej práci z dôvodu jednoduchšej použiteľnosti a potreby menšieho počtu pamäťových segmentov. Zvolenou metódou bilineárnej Z-transformácie, je popísaná aplikácia IIR filtrácie, pre jej možnosti využitia pri filtračnom riešení selektívnych filtrov použitých v realizácii nástroja. Kapitola sa venuje popisu typov selektívnych filtrov (kapitola 2.5.3) použitých v softvérovom nástroji a v poslednej časti 2.5.4 popisuje podrobný postup pri príkladnej ukážke návrhu filtra, ktorým je riešená programová algoritmizácia pri tvorbe softvérového nástroja využiteľného procesy filtrácie.

Zvukové signály je možné spracovávať rôznymi typmi procesov, medzi jeden z najpoužívanejších, ktorý bol využitý aj v konkrétnej práci, je proces filtrácie. Jedná sa o základné použitie procesu pri tvorbe zvuku, kedy vzniká výrazná statická alebo dynamická modifikácia pôvodného signálu.

Pod pojmom filter v jednoduchosti chápeme systém, ktorý selektívne mení žiadaným spôsobom vlastnosti zvukového signálu ako je jeho tvar, amplitúdovo-frekvenčná alebo fázovo-frekvenčná charakteristika. Základným využitím filtrácie je predovšetkým zvyšovanie kvality signálu (napríklad redukovaním šumu), pre extrakciu informácií zo signálu alebo jeho samotné oddelenie. [20]

2.5.1 Digitálny filter

Digitálny filter je matematický algoritmus implementovaný na hardvér alebo softvér, ktorého funkcia je založená na produkcii digitálneho výstupu z konkrétneho digitálneho vstupu. Digitálne filtre častokrát pracujú na zdigitalizovaných analógových signáloch, alebo len číslach, reprezentujúcich premenné uložené v pamäti počítača. [20]

Digitálna filtrácia zohráva veľmi dôležitú rolu pri digitálnom signálovom spracovaní (DSP). V porovnaní s analógovou filtráciou je preferovaná vo viacerých odvetviach ako sú dátové kompresie alebo zvukové či rečové spracovania, zároveň zabezpečuje viaceré výhody ako napríklad vytvorenie charakteristiky filtra, ktorú v analógových obvodoch nemožno dosiahnuť alebo možnosti zmeny charakteristiky bez zásahu do hardvéru. Dôležité je poznamenať, že digitálne spracovanie je nezávislé od teploty, vlhkosti, tlaku či iných vplyvov vonkajšieho prostredia v porovnaní s analógovým spracovaním. [20]

2.5.2 Typy digitálnej filtrácie: FIR a IIR filtre

Digitálne filtre je možné spravidla rozdeliť do dvoch tried v závislosti od dĺžky ich impulznej odozvy. Prvá trieda zahŕňa filtre, ktorých impulzná odozva nadobúda konečných hodnôt dĺžky – filtre triedy FIR (Finite Impulse Response). Tento typ filtrov je vždy stabilný a môže byť navrhnutý tak, aby mal presne lineárnu fázu. Do druhej triedy filtrov radíme všetky filtre, ktorých impulzná odozva nadobúda nekonečnej dĺžky – filtre typu IIR (Infinite Impulse Response). Filtre typu IIR nie sú vždy stabilné, a nenachádzame u nich lineárny fázový priebeh. [21]

V porovnaní filtrov IIR a FIR však nachádzame významnú výhodu v možnosti viac priblížiť špecifikovanú veľkosť odozvy pomocou filtra IIR oveľa nižšieho rádu, ako filtrom typu FIR. To znamená, že výpočtová zložitosť pri implementácii IIR filtra je oveľa menšia ako zložitosť filtra FIR na dosiahnutie rovnakých výsledkov. [21]

Celkový výber typu triedy filtrácie závisí na viacerých okolnostiach ako sú lineárny fázový priebeh, stabilita, ale aj výpočtová zložitosť či potrebná veľkosť pamäťovej kapacity. Spomenuté vlastnosti a charakteristiky pre obe triedy filtrov boli v pri realizácii tejto práce porovnávané a pre algoritmické riešenie digitálnej filtrácie bola zvolená trieda IIR filtrácie, z dôvodov alokovania menšieho počtu pamäťových premenných a z dôvodu použitia menej komplikovaných výpočtových funkcií. Zároveň si v návrhu uvedomujeme nedostatky tejto triedy filtrov.

Výber metódy pre návrh filtra triedy IIR

IIR filtre sú navrhnuté pomocou procesu mapovania z časovo-spojitej (analógovej) S-roviny do diskrétnej (digitálnej) Z-roviny. Spočiatku sa špecifikácie digitálneho filtra transformujú na špecifikácie analógového filtra v S-rovine. Analógový filter je potom spätne navrhnutý pred jeho transformáciou na digitálny filter v rovine Z. Výhoda priameho navrhovania analógového filtra namiesto digitálneho filtra spočíva hlavne v použití vyspelých a pokročilých poznatkov, ktoré sú pri návrhu analógových filtrov k dispozícii. Dôležitou požiadavkou pri procese mapovania je zabezpečiť, aby sa zachovali základné vlastnosti analógových filtrov - imaginárna os v analógovej S-rovine musí byť mapovaná do jednotkovej kružnice v rovine Z. Taktiež musí byť zachovaná stabilita. [21]

Pri navrhovaní filtrov IIR sa používajú dve metódy: *metóda impulznej invariancie* a *metóda bilineárnej z-transformácie*. Metóda impulznej invariancie zachováva impulznú odozvu analógového filtra. Veľkosť reakcie (magnitúda) však nie je zachovaná z dôvodu aliasingu. Metóda impulznej invariancie preto nie je dobrou metódou na navrhovanie digitálnych filtrov s vysokou priepustou alebo pásmovou priepustou či zádržou. Na druhej strane metóda bilineárnej z-transformácie je vhodná na zachovanie veľkosti reakcie, ale nie impulznej odozvy. Je preto vhodná na navrhovanie frekvenčne selektívnych filtrov. [21]

Metóda bilineárnej Z-transformácie

Vychádzajúc z teoretických poznatkov sme sa rozhodli pre aplikáciu filtrácie pomocou metódy bilineárnej Z- transformácie. Táto metóda mapuje prepojenie s-roviny a z-roviny pomocou substitúcie zobrazenej na matematickom vzťahu (2.2).

$$s = \frac{2}{T}\frac{z-1}{z+1} = \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}} \tag{2.2}$$

Koeficient T dostaneme pretočením (pre-warp) vlastností frekvencií u jednotlivých typov filtrov z digitálnej do analógovej roviny. [21]

Aby sa zamedzilo vzniku skreslenia na okraji pásma, musia byť tieto frekvencie deformované, pretočenie frekvenčných vlastností je realizované rovnicou (2.3). [21]

$$T = 2\tan\frac{2\pi\frac{F_c}{F_s}}{2} \tag{2.3}$$

F_s je samplovacia frekvencia a F_c je frekvencia závislá na typu filtra. [21] Viac v kapitole 2.5.3 Typy filtrov.

2.5.3 Typy filtrov

Ako bolo už v práci uvedené, poznáme viacero typov filtrov, ktoré sa líšia jednotlivými vlastnosťami. Následne budú predstavené typy filtrov ich vlastnosti a prenosové funkcie v analógovej rovine S.

Pod parametrom F_c rozumieme hraničnú frekvenciu (cutoff), charakterizovanú poklesom o 3dB oproti prechodnému pásmu. [23]

Pod parametrom Q rozumieme faktor kvality, charakterizovaný úbytkom alebo príbytkom amplitúdy (peaking) v oblasti frekvencie F_c. (Pri Butterworthovej aproximácii sa jedná o hodnotu Q = 0,707, ktorá je strednou hodnotou parametra Q.) [23]

Dolná priepusť / horná zádrž (LowPass/HiCut) – slúži na odfiltrovanie frekvencií vyšších ako F_c, respektíve prepustenie frekvencií pod frekvenciou F_c. [23]

Prenosová funkcia dolnej priepusti je realizovaná podľa rovnice (2.4).

$$H(s) = \frac{H_0}{s^2 + \frac{\omega_0}{Q}s + \omega_0^2} \tag{2.4}$$

Koeficient H_0 považujeme za zosilnenie (gain) a $\omega_0 = 2\pi F_c$. [23]

Horná priepusť / dolná zádrž (HighPass/LowCut) – slúži na odfiltrovanie frekvencií nižších ako F_c, respektíve prepustenie frekvencií nad frekvenciou F_c. (Opak dolnej priepusti.) [23]

Prenosová funkcia dolnej priepusti je realizovaná podľa rovnice (2.5).

$$H(s) = \frac{H_0 s^2}{s^2 + \frac{\omega_0}{Q}s + \omega_0^2} \tag{2.5}$$

Koeficient H_0 považujeme za zosilnenie (gain) a $\omega_0 = 2\pi F_c$. [23]

Pásmová priepusť (BandPass) – slúži na prepustenie frekvencií v oblasti frekvencie F_c v závislosti od parametru šírky pásma (Bandwidth) BW. [23]

Prenosová funkcia dolnej priepusti je realizovaná podľa rovnice (2.6).

$$\begin{aligned} H(s) &= \frac{H_0 \omega_0^2}{s^2 + \frac{\omega_0}{Q}s + \omega_0^2}, \\ Q &= \frac{F_c}{BW} = \frac{\sqrt{F_H F_L}}{F_H - F_L} \end{aligned} \tag{2.6}$$

F_H je horná frekvencia priepustného pásma a F_L je dolná frekvencia priepustného pásma. Ďalej $\omega_0 = 2\pi F_c = 2\pi\sqrt{F_H F_L}$ a kde H_0 je parameter zosilnenia. [23]

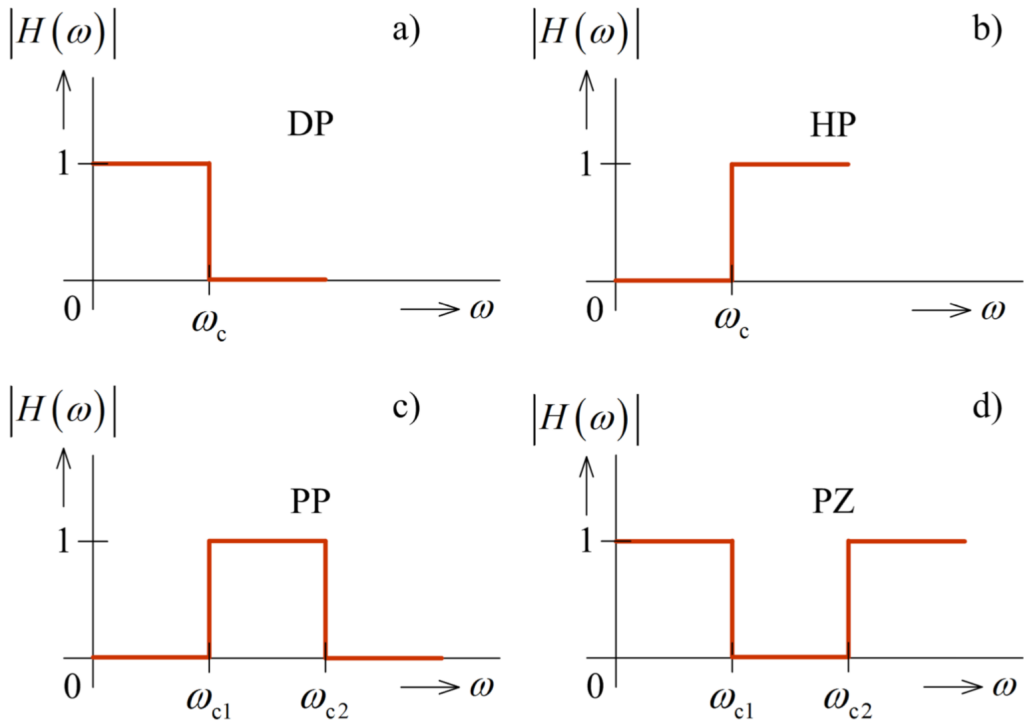
Pásmová zádrž (Notch) – slúži na odfiltrovanie frekvencií ležiacich v oblasti frekvencie F_c v závislosti od parametru šírky pásma (Bandwidth) BW. [23]

Prenosová funkcia dolnej priepusti je realizovaná podľa rovnice (2.7).

$$H(s) = \frac{H_0(s^2 + \omega_z^2)}{s^2 + \frac{\omega_0}{Q}s + \omega_0^2}, \qquad Q = \frac{F_c}{BW} = \frac{\sqrt{F_H F_L}}{F_H - F_L} \tag{2.7}$$

Parameter ω_z je uhlová frekvencia núl filtra a ω_0 je uhlová frekvencia pólov filtra, pre možnosť vytvorenia LowPass Notch alebo HighPass Notch filtra. Pri štandardnom a našom použití sa jedná o $\omega_z = \omega_0$.

Ďalej $\omega_0 = 2\pi F_c = 2\pi\sqrt{F_H F_L}$ a H_0 je parameter zosilnenia. [23]



Obr. 2.1: Modulové kmitočtové charakteristiky ideálnych filtrov: a) dolná priepusť (DP), b) horná priepusť (HP), c) pásmová priepusť (PP), d) pásmová zádrž (PZ).

Prevzaté z [22].

2.5.4 Návrh filtra

Ukážka navrhne IIR rekurzívny filter typu spodná priepusť (LowPass) s použitím bilineárnej transformácie pre výpočet koeficientov, ktoré budú dosadené do implementácie IIR filtra druhého rádu obsahujúceho dva póly a dve nuly, ktorý sa nazýva tiež bikvadratický filter (biquad).

Implementácia biquad IIR filtra je matematicky reprezentovaná obrazom (2.8).

$$H(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{b_0 + b_1 z^{-1} + b_2 z^{-2}} \tag{2.8}$$

Koeficienty a_i predstavujú nulové body filtra a koeficienty b_i predstavujú body pólov IIR filtra. [24]

Pre analogický zápis v diskrétnej digitálnej rovine je možné obraz prepísať podľa pravidiel z [22] do tvaru rovnice (2.9).

$$y[n] = \frac{1}{b_0}(a_0 x[n] + a_1 x[n-1] + a_2 x[n-2] - b_1 y[n-1] - b_2 y[n-2]) \tag{2.9}$$

Z rovnice (2.9) vyplýva, že koeficienty a_i a sú násobené so vstupnými hodnotami $x[n-k]$, a koeficienty b_i tvoria rekurzívnu časť filtru a sú násobené s výstupnými hodnotami filtru $y[n-k]$.

Prenosovú funkciu filtra spodnej priepusti podľa rovnice (2.4) je možné upraviť pri zvolení zosilnenia $H_0 = 1$ a $\omega_0 = 1$ na tvar rovnice (2.10).

$$H(s) = \frac{1}{s^2 + \frac{s}{Q} + 1} \tag{2.10}$$

Použitím bilineárnej Z-transformácie z S-roviny do Z-roviny podľa vzťahu (2.2) dostávame rovnicu obrazu (2.11).

$$H(z) = \frac{1}{(\frac{2}{T}\frac{z-1}{z+1})^2 + \frac{\frac{2}{T}\frac{z-1}{z+1}}{Q} + 1} \tag{2.11}$$

Pomocou matematických úprav prevedieme rovnicu obrazu (2.11) na tvar bikvadratického filtra reprezentovaného obrazom (2.8), z ktorého získavame koeficienty a_i a b_i.

Tieto koeficienty dosadíme do diskrétneho tvaru rovnice (2.9) spolu so vstupnými a výstupnými hodnotami signálu, ktorý filtrujeme. Pre každú vstupnú reprezentáciu digitálneho signálu, dostávame novú hodnotu signálu závislú na koeficientoch a_i a b_i, na vstupe konkrétnej informácie $x[n]$, na vstupe predošlých vzoriek signálu $x[n-k]$, časovo posunutých o hodnotu k, na ich momentálnom výstupe $y[n]$ a predošlých hodnotách výstupu $y[n-k]$ s posunom k.

Predvedeným spôsobom ukážky je možné realizovať rôzne typy filtrácie, závislé na prenosovej funkcii filtra.

2.6 Sonifikácia

Teoretické poznatky z nižšie uvedených informácií a problematík sonifikácie sú využité k vytvoreniu vhodnej algoritmizácie pri procese prevodu viditeľných parametrov farebných atribút do zvukovej podoby v konkrétnej práci.

Pod pojmom sonifikácia rozumieme typ techniky, ktorý sa využíva k prekladu informácií do zvukových dát a ich následné prepojenie. Sonifikácia je novodobo čoraz viac využívaný pojem v jednotlivých vedeckých odboroch, ako sú fyzika, akustika, psychoakustika, muzikológia alebo aj audioinžinerstvo a jej aplikácie nájdeme v témach, ako je napríklad teória chaosu, biomedicína, zariadenia pre nevidiacich alebo aj počítačové interakcie. [13]

Sonifikácia sa zameriava len na nehovorené zvuky, teda zvuky netvorené rečou človeka. [14]

2.6.1 Využiteľnosť a funkcionality sonifikácie

Prvotným zámerom sonifikácie je teda preložiť dáta, hocakých foriem a variant, do zvukového obrazu alebo naspäť. [13]

Dôvody tejto metódy prekladania sú napríklad upozorniť, varovať, monitorovať, získať dáta o procesoch a ich stavoch. Zároveň novodobo môžeme do dôvodu vzniku tohto prekladu zaradiť aj chápanie umenia, zábavy, športu či fyzického cvičenia. [15]

Pomocou sonifikácie dostávame preklad mimohudobných informácií do konkrétnejšej, zrozumiteľnejšej podoby, ktorá je úspešnejšia. [15]

2.6.2 Parametrické mapovanie

Jednou z viacerých sonifikačných techník je parametrické mapovanie určené k sonifikácii (PMSon). Je založené na premene parametrov informácií získaných z dát na parametre pracujúce s audio signálmi pre ich následné zobrazenie. Tento prenos je možný v prípade potreby prispôsobovať, aby v konečnom dôsledku bol adaptívny vlastnostiam vnímania a obmedzený podľa sluchových parametrov ľudského sluchu s cieľom optimalizácie interpretačného potenciálu. [13]

Dátová príprava

Jedným z krokov používania PMSon je objektívne vyhodnotenie údajov pre ich následné procesy, ako je dátová príprava. Dátová príprava je ovplyvňovaná aj zo strany štruktúry údajov, ale zároveň musíme brať v úvahu inherentné štruktúry dostupných parametrov pre následné použitie pripravených dát, pre syntézy. [13]

Použitie a dôvody využitia PMSon

Pre použitie metódy PMSon sú potrebné nasledujúce kroky: vyhodnotenie vstupných údajov, príprava dát, prenosový proces, mapovanie k výstupu a reflexia s následnou spätnou väzbou. [13]

Najprv sa zameriavame na počúvanie výsledného signálu vzniknutého z audio parametrov, komplexne ho riešime v časovej rovine, zameriavame sa na informatívnosť a vernosť prekladu. Zároveň ho riešime po technickej stránke z hľadiska, ako je čistota zvuku, jeho možné meškanie či výpadky. Nakoniec signál ladíme, kedy po vyhodnotení dát zvukovodom metódu ovplyvňujeme, vylepšujeme, zdokonaľujeme. Využitie PMSon v nadobúda v dnešnej dobe široký význam vzhľadom na vysokú efektívnosť prekladu dát z rôznych fyzických alebo logických dát do zvukovej oblasti. [13]

Pomocou sonifikácie získavame preklad mimohudobných informácií do konkrétnejšej, zrozumiteľnejšej podoby. [15]

2.7 Dátové formáty

Dôvodom uvedenia kapitoly 2.7 je výsledok činnosti softvérového nástroja, ktorým je zvuková ukážka vytvorená typom zvukového formátu.

Kapitola sa sústreďuje na objasnenie problematiky z oblasti dátových formátov a upresňuje formát pre vytvorenie alebo uloženie digitálnych zvukových dát. V dnešnej dobe je možno využiť veľkého množstva zvukových formátov, rozdelených podľa kategórií súvisiacich s ich využitím.

Požitý zvukový formát WAVE, bližšie popísaný v kapitole 2.7.2, bol vybraný na základe použiteľnosti na rôznych počítačových platformách. Zároveň sa jedná o bezstratový zvukový formát (dáta ukladá bez stratových kompresií) v porovnaní s formátmi, ktoré kompresiám podliehajú napríklad: MP3, WMA alebo FLAC. Taktiež je jeden z najznámejších a najpoužívanejších formátov v porovnaní s menej známymi ako napríklad formáty typu AU alebo AIFF.

Dlhodobým špecifikom je úspešnosť stratégie ukladania súborov digitálneho audia na diskové oddiely, ktoré boli v minulosti pomerne pomalé a ťažko zabezpečovali väčší zvukový záznam údajov. Nedávny nárast významu metadát (informácie o dátach) viedol k vývoju metódy vytvárania formátov obsahujúcich zvukové údaje ako objektovo orientované koncepty. [25]

Kapitola sa zaoberá špecifikáciou zvukového formátu, ktorý bol využitý pri realizácií softvérového nástroja.

2.7.1 Základné informácie

Pod pojmom dátový formát možno chápať rad dátových bajtov formovaných do blokov a uložených v súvislej alebo vo fragmentovej podobe. V istom zmysle sú samotné súbory nezávislé od operačného systému a štruktúry archivovania hostiteľského počítača, preto je možné súbor preniesť na inú platformu so stále existujúcou sériou dátových blokov. [25]

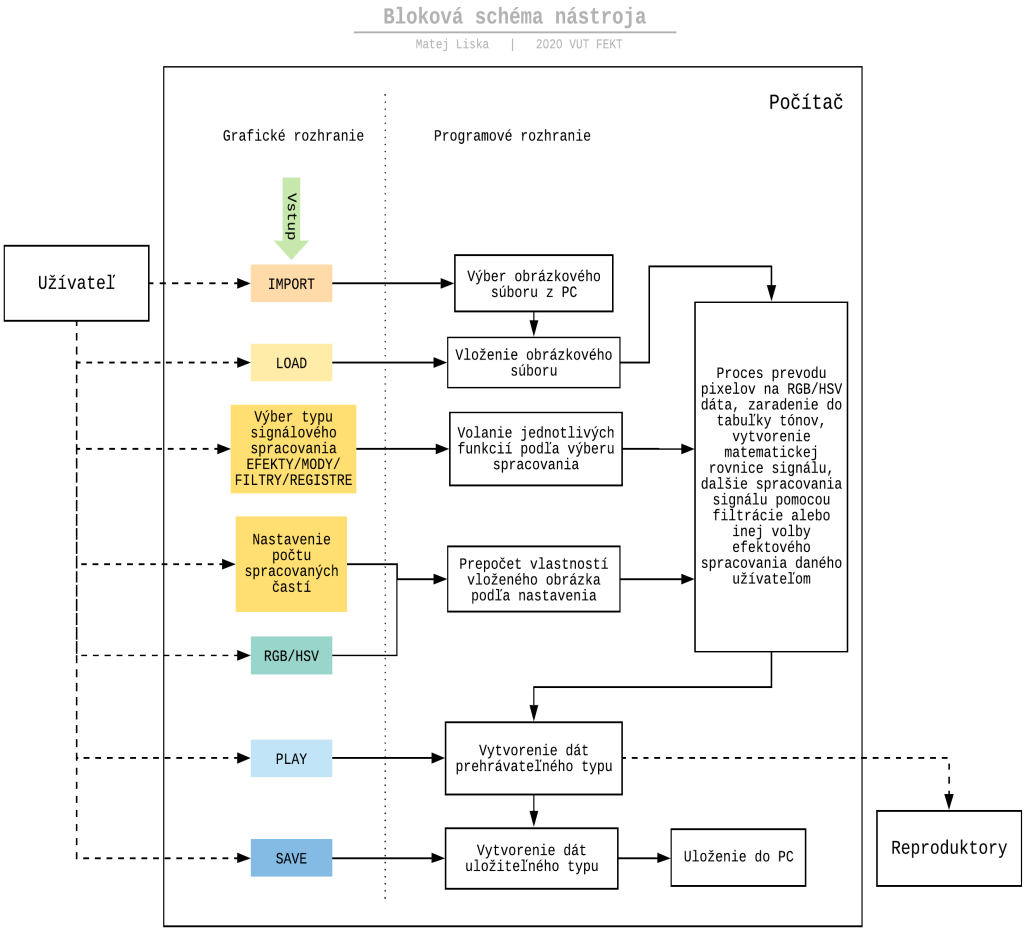
Existujú dva prístupy k usporiadaniu bajtov: formát s little-endian poradím, v ktorom je najmenej významný bajt na prvom mieste alebo na adrese s najnižšou pamäťou, a formát big-endian, v ktorom najvýznamnejší bajt je na prvom mieste alebo na najvyššej adrese pamäte. [25]

2.7.2 Zvukový dátový formát RIFF WAVE

Formát RIFF WAVE (často nazývaný WAV) sa používa všeobecne na spracovanie zvukových súborov. Jedná sa o bezstratový zvukový formát bez kompresie dát. Formát môže byť uložený typom little-endian aj big-endian. Základný súbor WAV sa skladá z troch hlavných častí: RIFF, FORMAT a DATA. [25] V časti RIFF sa nachádzajú metadáta vrátane informácií o celkovej dĺžke, type formátu alebo type modulácie. U wav sa jedná o PCM (pulzná kódová modulácia). Taktiež je vložená informácia o počte kanálov (MONO = 1, STEREO = 2), pre ktoré boli dáta generované. V časti FORMAT sa nachádzajú informácie popisujúce zvolený formát, vzorkovací kmitočet, počet bitov za sekundu, počet bitov za jeden sampel. V poslednej časti DATA sa nachádzajú samotné vygenerované dáta určené pre vytvorenie zvukového formátu. [25]

Zvukové vzorky sa vkladajú podľa kanálov v časovom poradí, takže ak súbor obsahuje dva kanály, za vzorkou pre ľavý kanál nasleduje bezprostredne priradená vzorka pre pravý kanál. [25]

3 Návrh



Obr. 3.1: Bloková schéma nástroja.

Vytvorené pomocou [16].

Bloková schéma názorne zobrazuje zjednodušený model nástroja, jednotlivé bloky a komunikáciu medzi nimi. Užívateľovi je umožnený výber obrázka, jeho následné vloženie a úpravy pomocou grafického rozhrania. Taktiež ovláda voľbu typu mapovania farieb, signálové spracovanie obrázka a možnosť rozdelenia obrázka do vybraných častí a ich jednotlivé úpravy, spracovania. Nakoniec je možné výsledný zvukový materiál prehrať pomocou reproduktorov alebo ho uložiť do počítača.

Vstupná informácia je vkladaná pomocou tlačidla IMPORT z pamäte počítača a načítaná pomocou tlačidla LOAD, kedy sa dostáva do hlavnej programovej časti. V tejto časti je importovaný obrázkový materiál spracovaný do parametrov RGB a prevedený na hodnoty amplitúd a frekvencií. Ďalej sú informácie spracovávané individuálne podľa zvolených úprav užívateľom. Tlačidlami sú ovládané typy signálového spracovania použitím efektov, filtrácie alebo rozdelenia obrázka na samostatne spracovateľné časti. Bloková schéma pre zjednodušenie vysvetlenia funkčnosti softvérového nástroja neobsahuje znázornené konkrétne typy a možnosti zvukového spracovania. Ako príklad môžeme uviesť spracovanie filtráciou, zapnutím funkcie pre dolnú priepusť - HiCut filter, alebo využitie ponúkaných efektov nástroja popísaných bližšie v kapitole 4 - Realizácia. Možnosťou rozdelenia vstupného obrázkového súboru je užívateľovi ponúkaný výber vytvorenia dvoch, štyroch či ôsmich častí, ktoré je možné jednotlivými voľbami efektového spracovania upravovať a úpravy kombinovať individuálne, pri každej zvolenej časti. Nastavenia volieb spracovania sú ukladané u každej časti obrázka do pamäte programu a po zmene výberu upravovanej časti na inú a následnom vrátení sa, je možné pokračovať v predošlom nastavení kontinuálne. Taktiež má užívateľ možnosť zmeniť mapovanie obrázka z RGB do HSV parametrov farieb, pre porovnanie zvukových vlastností. Výsledný zvukový materiál je možný prehrať pomocou tlačidla PLAY kedykoľvek po kroku načítania (LOAD) alebo uložiť vo formáte Waveform (.wav) na vlastný počítač pomocou tlačidla SAVE, kedy je potrebné zadať názov, pod ktorým bude ukážka uložená. Vložený obrázkový materiál je po načítaní (LOAD) zobrazený v softvérovom rozhraní grafickej časti.

Bloková schéma vo výslednom tvare zobrazuje všetky základné bloky potrebné k vytvoreniu požadovaného nástroja. Typy signálového spracovania sú zoskupené v jednom bloku pre zjednodušenie schémy a zároveň je takéto rozdelenie a rozloženie blokov použiteľné aj po pridaní nových, iných typov signálového spracovania. Grafická časť spolupracuje s programovým rozhraním použitím jazyka FXML a Java.

4 Realizácia

Kapitola Realizácia je zameraná na zobrazenie a popis funkčnosti grafického návrhu softvérového zariadenia a vysvetlenie programového riešenia jednotlivých častí zariadenia vrátane kódových ukážok. Popisuje ich technický návrh, štruktúru prepojenia a ich vlastné či vzájomné funkčnosti. Po priblížení základných informácií a návrhu užívateľského grafického rozhrania vrátane jeho ovládania, kapitola popisuje vkladanie a spracovanie obrázkového súboru, možnosti delenia obrázkových súborov, popisuje mapovanie farebných atribút do dvoch rôznych modelov farieb, popisuje generovanie syntetických zvukov použitím matematických algoritmov, tvorbu prehrávateľného a uložiteľného formátu zvukových súborov, zobrazuje dve možnosti signálového spracovania pri vytváraní zvukových dát alebo pri následnej úprave vytvorených dát.

Okrem toho sa kapitola venuje vysvetleniu princípu algoritmizácie jednotlivých častí programu slovným aj obrázkovým popisom problematiky pred ich implementáciou do kódového riešenia. Z dôvodu vytvorenia plynulého chodu a zamedzeniu prípadnému pádu softvérového zariadenia sú vysvetlené použitia pomocných operácií či premenných implementovaných v kóde, zabezpečujúcich stabilitu zariadenia.

Realizácia softvérového zariadenia vychádza z teoretických podkladov a poznatkov uvedených v tejto práci. Softvérové zariadenie je riešené programovacím jazykom Java bez použitia externých knižníc.

4.1 Základné informácie

Softvérové zariadenie experimentálneho zvukového nástroja riadeného farbami svetla sprístupňuje užívateľovi možnosť previesť obrázkový súbor na zvukové médium. Hlavnou funkciou nástroja je vytvoriť zvukovú ukážku pomocou tabuľky vychádzajúcej z možnosti prepočtu svetelného spektra na zvukové. Z jednotlivých pixelov sú vytvorené farebné atribúty RGB alebo HSV, použité pre generovanie a spájanie zvukových signálov. Vytvorený zvukový signál je možné upravovať jednotlivými signálovými procesmi podľa voľby užívateľa. Signálové procesy je možné medzi sebou kombinovať. Nástroj má k dispozícii možnosť ovládania výberu počtu častí, na ktoré vstupný obrázkový materiál rozdelí a je možné pracovať s každým segmentom obrázka individuálne. Týmto rozdelením užívateľ nastavuje príslušné signálové procesy nezávisle každému segmentu zvlášť. Každé nastavenie je v zariadení ukladané a vo výsledku je možné spätne upravovať segmenty viackrát.

Výsledný materiál syntetického zvuku je umožnené prehrať alebo ukladať do počítača užívateľa pre využitie ako napríklad samplu, resp. časti zvuku do hudobnej skladby, počítačových hier, zvukových efektov.

4.1.1 Programovacia platforma JavaFX s grafickým rozhraním

Softvérová realizácia experimentálneho hudobného nástroja je v dnešnej dobe riešiteľná viacerými programovacími spôsobmi. Je umožnené riešiť komplikované softvérové operácie, týkajúce sa tvorby alebo spracovania zvukových signálov pomocou jednoduchších programovacích jazykov, editorov a programovacích prostredí, cez stredne obťažné programovanie, až po programovanie vlastného typu funkcií, všetkých štruktúr, tried, objektov či premenných.

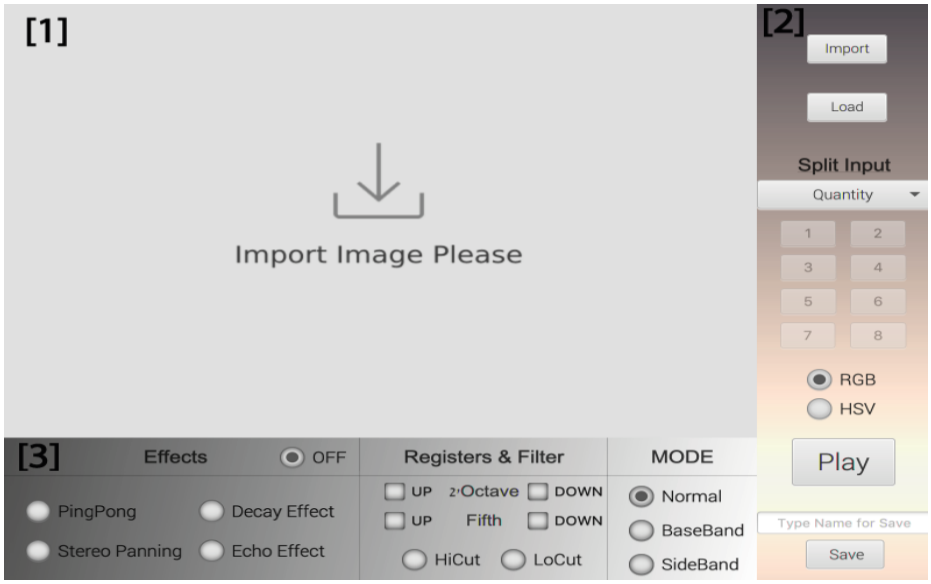
K výberu programovacieho jazyka sme sa snažili pristupovať profesionálnym spôsobom a boli vyradené možnosti ponúkané jednoduchými grafickými programovacími editormi, kedy sa program netvorí písaním vlastného kódu, ale zadávaním vlastností či hodnôt do vopred vytvorených funkcií ponúkaných editormi, kde prepojenia boli riešené potiahnutím jednotlivých predprogramovaných blokov kódu pomocou kurzoru a ich vnútorné algoritmizácie nebolo možné individuálne meniť. K tvorbe našej práce sme vyberali medzi najpoužívanejších programovacími jazykmi a bol zvolený jazyk Java, v ktorom je umožnené a potrebné vytvárať celý vlastný kód.

JavaFX je programovacia platforma, ktorá využíva programovací jazyk typu Java a je obohatená o grafické rozhranie, ktoré je možno tvoriť pomocou programovacieho užívateľského rozhrania FXML, založeného na značkovacom (markup) jazyku XML.

Pre využitie programovacej platformy JavaFX a k vytvoreniu programového riešenia práce boli potrebné základné poznatky z programovania. Základmi programovania sa táto práca nezaoberá.

4.2 Grafický dizajn užívateľského rozhrania

Vytvorenie grafického užívateľského prostredia (ďalej len GUI) nástroja vychádzalo z návrhu blokovej schémy znázornenej na obr. 3.1, a je zobrazené na obr. 4.1.



Obr. 4.1: Grafické užívateľské prostredie.

Prvotným zámerom (GUI) nástroja bolo vytvoriť jednoduché a intuitívne prostredie s množstvom ovládacích prvkov a funkčným vzhľadom. Preto sme zvolili vizuálne rozdelenie nástroja na tri základné časti.

V prvej časti (označená ako 1 na obr. 4.1) sa po importovaní zobrazí vložené obrázkové médium. Dôvodom je nevyhnutnosť informovať užívateľa o vzhľade vstupnej zložky, s ktorou pracuje a zároveň je dôležitý celkový audiovizuálny výsledok nástroja. Druhá časť nástroja (označená ako 2 na obr. 4.1) pracuje prevažne s vizuálnou zložkou nástroja. Nachádzajú sa v nej tlačidlá pre výber a načítanie obrázkového média, pre možnosť rozdelenia obrázka do viacerých častí, výber mapovania vstupu do dvoch rôznych farebných akronym (RGB, HSV) a možnosti prehrať alebo uložiť vytvorenú zvukovú ukážku pod menom zvoleným užívateľom. V poslednej tretej časti (označená ako 3 na obr. 4.1) je možné pracovať výhradne so zvukovou zložkou nástroja. Jednotlivými nastaveniami je možné upravovať vytvorený audio materiál pomocou efektov, registrov alebo filtrácie.

Vytvorené rozdelenie celkového nástroja týmto spôsobom umožňuje rýchlejšiu, intuitívnejšiu a efektívnejšiu orientáciu pri jeho používaní. Pre flexibilnosť využitia nástroja bol pri tvorbe GUI zvolený anglický jazyk, dôvodom je širšia verejná použiteľnosť. GUI bolo vytvorené jazykom FXML, ktorý obsahuje platforma JavaFX.

4.3 Vkladanie obrázka

Po spustení softvéru je prvým krokom výber a vloženie obrázka pre vytvorenie zvukového materiálu a jeho spracovanie. Výber obrázkového súboru je realizovaný tlačidlom IMPORT, ktorým je volaná funkcia pressImport, funkcia automaticky otvorí užívateľovi prehliadač súborov na diskoch počítača a umožní výber obrázka. Po zvolení sa zobrazí absolútna cesta k vybranému médiu v textovom kontajneri *URLpath*, nachádzajúcim sa pod tlačidlom IMPORT, pre možnosť overenia pred načítaním vybraného média do softvérového nástroja. Celý proces vytvorenia cesty je zároveň kontrolovaný podmienkou, overujúcou, že vybratie cesty užívateľom bolo uskutočnené.

Kódové riešenie výberu na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.1: Výber cesty pomocou tlačidla IMPORT.

```
public void pressImport(ActionEvent event){
    File file = fileChooser.showOpenDialogog(null); //Otvorenie okna k vyberu z PC
    if(file != null){                //Podmienka ak uzivatel vyberie subor
       path = file.getAbsolutePath(); //Ziskanie cesty pre otvorenie obrazka
       URLpath.setVisible(true);      //Zviditelnenie textoveho kontajneru
       URLpath.setText(path);         //Vlozenie a zobrazenie vybranej cesty
}
```

Tlačidlom LOAD sa výber cesty obrázka skontroluje a načíta do programu. Po stlačení LOAD je pre overenie kontrolovaná prípona vloženého súboru pomocou funkcie *checkExtension*, z dôvodu predídenia prípadným chybám pri vkladaní od užívateľa. Podporované sú prípony jpg, png a tif. V prípade vloženia iného typu prípony je zobrazená plocha uložená v programe nástroja oznamujúca chybu. Po kontrole prípony sa vložený obrázok zobrazí na GUI nástroja, a zároveň sú funkciou *convert* s vloženým obrázkom prevedené a rozdelené informácie do premennej *convertRGB*, keďže základné nastavenie pracuje s mapovaním do RGB akronym. Ďalším krokom je vytvorenie audio dát z RGB informácií pomocou triedy *SynGen* a funkcie v nej. Funkcie a trieda sú podrobnejšie popísané v kapitolách 4.4 a 4.5.

Kódové riešenie načítania na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.2: Načítanie do softvéru tlačidlom LOAD.

```
public void pressLoad(ActionEvent event) throws IOException {
if(path != null){
       FileInputStream inputStream; //premenna do ktorej sa nacita cesta obrazka
       if(checkExtension(path)){   //podmienka pre skontrolovanie pripony
    inputStream = new FileInputStream(path); //vlozenie cesty k nacitaniu obrazka do programu
}else{ inputStream = new FileInputStream("src/sample/Drawable/Oops.png"); //error vypis uzivatelovi
 }
BufferedImage imageReal = ImageIO.read(new File(path)); //nacitanie obrazka
CalculateRGB = convert(imageReal, hsv);//premenna pola: CalculateRGB volajúca funkciu convert()
new SynGen().getSynthesizData(audioData, CalculateRGB);//volanie triedy pre vytvorenie zvuk. dát
}
```

4.4 Získanie a zoradenie informácie z pixelov

Proces získavania informácií z parametrov uložených v pixeloch je v našej práci využitý po vložení, rozdelení alebo výbere konkrétnej časti obrázkového súboru, preto bolo potrebné vytvoriť funkciu s návratovou hodnotou a možnosťou volania z viacerých miest programu. Funkciu zabezpečujúcu proces sme nazvali *convert*. Pri jej použití je potrebné deklarovať premennú, do ktorej sú získané informácie vložené (príklad *CalculateRGB*). Premenná bude inicializovaná vrátenými hodnotami funkcie *convert*, s vloženým obrázok (napríklad *imageReal*). Premennú zadeklarujeme ako pole. Podľa voľby mapovania do RGB alebo HSV akronym je funkcia *convert* volaná zároveň s premennou hsv, označujúcou typ mapovania. Premenná hsv je typu boolean.

Kódové riešenie na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.3: Deklarácia premennej volajúcej konvertujúcu funkciu.

```
//premenna pola CalculateRGB volajúca funkciu convert() kde je vložený obrázok
int [] CalculateRGB = convert(imageReal, hsv);
```

Funkcia *convert* z nahratého obrázku získa vlastnosti jeho šírky a výšky. Pomocou cyklov a vlastností obrázkovej veľkosti prechádza jednotlivo pixel po pixeli, z ktorých získava RGB parametre do premennej *color* funkciou *getRGB* obsiahnutej v JavaAPI rozhraní. Parametre posiela do rozdeľujúcej funkcie *convertToScale*, vracajúcej číselné hodnoty označujúce jednotlivé farby podľa obr. 4.2 a realizujeme rozdelenie do 14 možných farieb. Premennú *result* deklarujeme ako pole o veľkosti 14, pole tiež čistíme, aby sa predišlo prípadným kauzám v programe. Do rozdeľujúcej funkcie je taktiež predávaná hodnota hsv. Funkcia podľa hsv hodnoty rieši dva typy mapovania, kedy sú parametre RGB prevedené do akronym HSV funkciou *RGBtoHSV* v JavaAPI rozhraní.

Kódové riešenie na nasledujúcej ukážke naprogramované autorom práce.

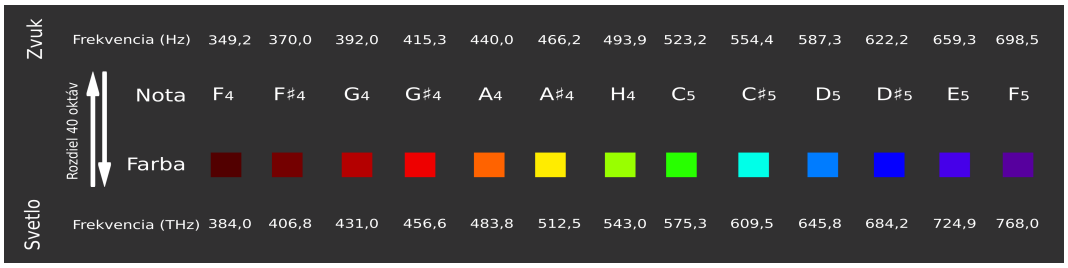
Výpis 4.4: Funkcia získania frekvencie, amplitúdy a počtu tónov.

```
private static int []convert(BufferedImage image, boolean hsv){
    int width = image.getWidth(); //ziskanie šírky obrazku
    int height = image.getHeight(); //ziskanie výšky obrazku
    int [] result = new int [14]; //pole pre ukladanie vysledkov
    for(int i = 0; i < 14 ; i++){ result[i] = 0; } // čistenie pola
    //cyklus zabezpečujúci získanie RGB z každého pixela a zaradenia do farby
    for (int row = 0; row < height; row++) {
    for (int col = 0; col < width; col++) {
    //získanie hodnot RGB z farby pixelu (možnost konvertovat do HSV)
    Color color = new Color(image.getRGB(col, row));
    //ziskanie číselnej hodnoty z tabulky podla farieb a typu mapovania RGB aleboHSV
    int number = convertToScale(color, hsv);
    result[number]++; }} //pričitanie jednotky u hodnoty farby
return result; } //vratenie finalneho zoradenia tónov podla RGB alebo HSV hodnoty
```

Pri modeli RGB parametrom R označujeme farbu červenú, B farbu modrú a G je zelená farba. Pri modeli HSV parameter H označuje odtieň, S je parametrom sýtosti a V označuje jas pixelu. Získanými parametrami sa pixel zaradí do tabuľky vychádzajúcej z obr. 4.2 pre zvolený typ mapovania funkciou *convertToScale*, viac v kapitole 4.4.3.

Rozdelenie, do ktorého je farebné zloženie pixelu zaradené, vychádza z teoretického poznatku o svetelnom spektre, ktoré sa nachádza približne o 40 oktáv vyššie ako počuteľné. Popísaným princípom je možné spracovať a preložiť každú farbu zo svetelného vlnenia do pásma zvukového. Šírka svetelného vlnenia je podrobnejšie popísaná v kapitole 1.1.2 a jej problematika rozloženia spektra do farieb v kapitole 1.2.3.

Podľa frekvenčnej šírky svetelného spektra a chromatickej postupnosti frekvencií zvukových tónov je možné zostaviť rozloženie do trinástich farebných hodnôt. Frekvenčne v zvukovom spektre dostávame tóny od F4 po F5.



Obr. 4.2: Prepočet frekvencií počuteľného a viditeľného spektra. [17]

Tóny F4 a F5 možno považovať za rovnaké z dôvodu ich oktávovej vzdialenosti. Ich spojením do jednej hodnoty a následným rozšírením o hodnoty označujúce farbu čiernu a bielu získavame 14 finálnych označení, do ktorých budeme farby radiť podľa voľby typu mapovania RGB alebo HSV akronym.

4.4.1 Mapovanie do RGB akronym

Každá hodnota R, G alebo B pixelu môže nadobudnúť hodnôt v rozmedzí od 0 do 255, kde 0 značí minimum a 255 maximum použitej zložky farby. Ak sú všetky hodnoty pixelu menšie ako 51, dostávame sa k farbe čiernej, ak sú všetky tri hodnoty vyššie ako 204, dostávame sa k bielej farbe. Pomocou týchto farieb realizujeme pridávanie oktáv.

Posudzovanie bude nasledovné: ak sa na obrázku bude nachádzať viac ako 50% tmavých pixelov (hodnoty RGB menšie ako 51), tóny, ktoré sa na obrázku nachádzajú, sa skopírujú a s polovičnou amplitúdou prenesú frekvenčne o oktávu nižšie,

ak sa bude nachádzať prevažne biela farba (RGB väčšie ako 204), tóny budú skopírované o oktávu vyššie. Týmto spôsobom sa snažíme dosiahnuť rozšírenia výsledného spektra zvuku, keďže 12 hodnôt sínusových signálov nevytvára zvukovo zaujímavé vzorky.

Taktiež poznamenávame, že pri importovaní čisto bieleho alebo čierneho obrázka, či obrázka so zložkami šedej škály by sme nedostali žiadny zvukový výstup. Táto kauza je riešená v prípade nášho nástroja frekvenciou tónu orchestrálneho A, ktoré nadobúda frekvenciu 440 hertzov. Znova podľa rozloženia počtu tmavšieho alebo bledšieho odtieňa šedej dostávame oktávy vyššie alebo nižšie od základnej frekvencie 440 hertz.

Výber tejto frekvencie bol realizovaný z technického hľadiska, ktoré sa opiera o skutočnosť, že túto frekvenciu je možné produkovať veľkou väčšinou reproduktorov, z hľadiska voči sluchovému ústrojenstvu, že táto frekvencia je dobre počuteľná a z hľadiska hudobného, frekvencia, na ktorú sa ladia hudobné nástroje.

Rozdelenie ostatných farieb vychádzalo z obr. 4.2. Problematiku mimo spektrálnych farieb popísanú v kapitole 1.2.3 sme riešili rozšírením frekvenčnej oblasti tónov, ktoré sa nachádzajú pri týchto farbách a vytvorili sme tabuľku zobrazenú na obr. 4.3, ktorá je vytvorená rozložením jednotlivých farebných hodnôt do tónov.

F	F	A♯	H	C	C	C♯	D	E	F	F	F
F♯	F♯	A♯	H	C	C	C♯	D	E	F	F	F♯
G	G	A♯	H	C	C	C♯	D	E	F	F	G
G♯	A	A♯	H	C	C	C♯	D	D♯	F	F	G♯
G♯	A	A♯	H	C	C	C♯	D	D♯	F	F	G♯
G♯	A	A♯	H	C	C	C♯	D	D♯	F	F	G♯
G♯	A	A♯	H	C	C	C♯	D	D♯	F	F	G♯

Obr. 4.3: Tónové obsadenie jednotlivých farieb pre RGB mapovanie. [18]

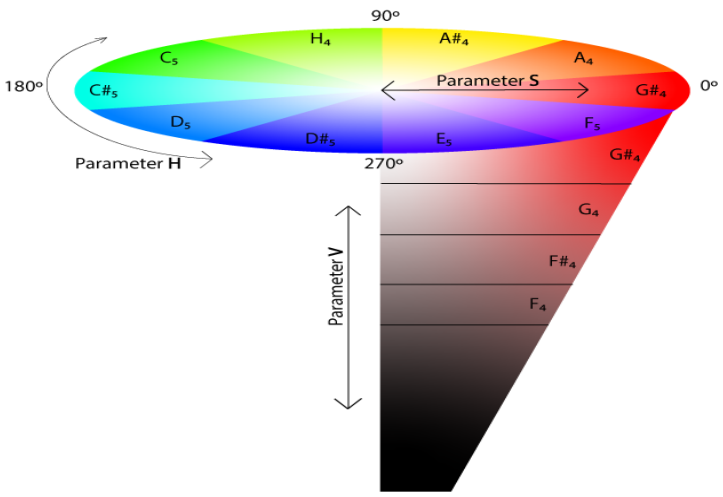
V konečnom dôsledku je možné dostať 14 hodnôt, z ktorých je 12 farebných zoradených podľa obr. 4.3 a zostávajúce dve sú farby čierna a biela, v prípade farebného vloženého obrázka. Programovo boli tónové rozmedzia riešené pomocou vnorených podmienok, kedy sme prechádzali medzi farbami na obr. 4.3 z ľavej strany na pravú a hranice tvorila nasledovná farba. Kódové riešenie je popísané v kapitole 4.4.3.

4.4.2 Mapovanie do HSV akronym

Model HSV vychádza z obsadenia farebnej škály v kruhovom tvare, kde je farbám priradená hodnota H v rozmedzí stupňov 0 až 360 a ostatné dve hodnoty S a V nadobúdajú percentuálne vlastnosti v rozmedzí od 0 do 100. Rovnakým princípom ako pri modeli RGB realizujeme pridávanie oktáv pomocou farieb bielej a čiernej, ktoré sú nezávislé na hodnote H. Ak parameter S nadobúda hodnoty 0 a V je menší ako 20%, jedná sa o farbu čiernu. Ak je parameter S menší ako 20% a zároveň je parameter V s hodnotou 100%, jedná sa o farbu bielu.

Podobne ako pri modeli RGB aj model HSV rieši ostatné farby podmienkovou štruktúrou a rozšírením zvukovej frekvenčnej oblasti vychádzajúcej z obr. 4.2.

V prvých štyroch farbách, respektíve tónoch F4 až Gis4, sa podľa hodnôt HSV vychádzajúcich z obr. 4.2 nemenil parameter H, označujúci odtieň farby, nastavili sme ho staticky pre tieto 4 hodnoty a delenie prebiehalo podľa parametra V, ktorý stúpal až po hodnotu 100. Parameter S nebol braný do úvahy, z dôvodu rozšírenia spektra nezávislého na saturácii. Ostatné farby boli závislé len na odtieni (parameter H) farieb a ich rozdelenie vychádzalo z obr. 4.2 od najnižšej frekvencie po najvyššiu. Všetky hodnoty parametra H nižšie ako hodnota H ďalšej farby patria do predchádzajúcej farby, nezávisle na parametroch S a V, ktoré neurčujú farebný odtieň. Pri porovnaní modelov RGB a HSV, bolo konštrukčne jednoduchšie mapovanie do modelu HSV, keďže farebný odtieň je riešený len parametrom H a nebolo potrebné prepočítavať farby na parametre trikolóru RGB. Výsledné tónové obsadenie vo farbách je znázornené na obr. 4.4.



Obr. 4.4: Tónové obsadenie v jednotlivých farbách pri HSV modeli.

[vytvorené autorom práce]

Ako pri modeli RGB je možné dostať 14 hodnôt, z ktorých je 12 farebných zoradených podľa obr. 4.4, keďže tón F4 a F5 znova považujeme za oktávu vzdialený rovnaký tón a zostávajúce dve sú farby čierna a biela, v prípade farebného vloženého obrázkového média. Kódové riešenie je popísané v kapitole 4.4.3.

4.4.3 Kódové zoradenie tónov do farieb podľa typu mapovania

Rozložením popísaným v kapitolách 4.4.1 a 4.4.2 sme vytvorili vlastný návrh prípravy dát pre dva rozdielne typy mapovania farieb do tónov pomocou teórie techniky sonifikácie, ktorá je uvedená v kapitole 2.6. Funkciu, ktorá bude vykonávať rozdelenie, nazývame *convertToScale* a nadväzuje na výpis kódu 4.4. Do funkcie vstupujú hodnoty farieb každého pixela a premenná označujúca typ mapovania.

Funkcia *convertToScale* s návratovou hodnotou vracia číslo označujúce konkrétnu farbu podľa typu mapovania. Informácie sú delené podmienkovou štruktúrou.

Kódové riešenie na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.5: Funkcia rozdelenia farieb do tónu podľa typu mapovania.

```
private static int convertToScale(@NotNull Color Value, boolean hsv){
    int B = Value.getBlue(); //hodnota modrej farby
    int G = Value.getGreen();//hodnota zelenej farby
    int R = Value.getRed();//hodnota cervenej farby

    if(hsv){ //premenna hsv oznacujuca typ mapovania (ak je true = mapovanie hsv)
        //HSV mapovanie
        float[] hSv = new float[3];
        Color.RGBtoHSB(R,G,B,hSv); //Konvertovanie z RGB do HSV akronym

       if (hSv[1] == 0 || hSv[2] < 0.2 || (hSv[1] < 0.2 && hSv[2] == 1)){
           //Black And White
           if(hSv[2] == 1){
               return 1; //white (biela)
           }else return 0; //black (cierna)
       //other colors (ostatne farby)
       }else if(hSv[2] < 0.32 && hSv[0] == 0){
           return 2; //F tone
       }else if(hSv[2] < 0.45 && hSv[0] == 0){
           return 3; //F+ tone
    //atd. dalsie farby
    }else{
      //RGB mapovanie
      //Riesenie odtienov sedej
      if((R == G && G == B) || (R <= 51 && G <= 51 && B <= 51) ||
            (R >= 204 && G >= 204 && B >= 204)){
        if (R <= 128)return 0; //black (cierna)
        else return 1; } //white (biela)
      //other colors (ostatne farby)
      if(R > 51 && R <= 102 && G <= 51 && B <= 51)return 2; //F TONE
      else if(R > 102 && R <= 153 && G <= 76 && B <= 76)return 3; //F+ TONE
    //atd. dalsie farby
    }
}
```

Po výbere zaradenia farby do tónu sa v programovom riešení zväčší hodnota vo vybranej farbe o jednotku. Takýmto spôsobom sa získava kompletná tabuľka, v ktorej nachádzame zaradené farby do tónov, počet vyskytnutých farieb a počet obsadenia jednotlivej farby vo vloženom obrázkovom médiu. Ich počet obsadenia považujeme za percentuálne vyjadrenie vo vstupnom obrázkovom médiu a pri konverzii na zvukové médium ho považujeme za amplitúdu. V konečnom dôsledku dostávame frekvenčnú zložku z tabuľkového rozloženia a amplitúdovú zložku podľa percentuálneho výskytu farby na vstupnom obrázkovom médiu.

V programovom riešení je do vytvorenej premennej (výpis kódu 4.3 premenná *CalculateRGB*) vkladané pole obsahujúce jednotlivé farby, získané z funkcie *convertToScale* (výpis kódu 4.5) a početnosť výskytu jednotlivých farieb v poli zároveň s celým riadením procesu je zabezpečené funkciou *convert* (výpis kódu 4.4).

4.5 Vytvorenie zvukového materiálu

Parametrami farieb, ktoré boli získané a rozdelené predošlými funkciami a s počtom ich výskytu je umožnené jednotlivé informácie previesť na zvukový formát.

Vytvorenie prehrávateľného a uložiteľného typu dát realizuje v tejto práci trieda *SynGen*, ktorá obsahuje funkciu *getSynthesizData*, do ktorej vkladáme parametre vytvorené a pripravené z obrázkového média a pole, do ktorého budú vytvorené dáta zvukového formátu spätne vložené. Triedu vrátane funkcie využívame z viacerých miest podľa volieb užívateľa, hneď pri stisnutí tlačidla LOAD, kedy sa vytvorí zvuková stopa len z informácií získaných z pixelov, pri každej voľbe signálového spracovania, filtrácie alebo pri rozdelení obrázkového vstupu do viacerých častí.

Kódové riešenie na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.6: Volanie funkcie pre vytvorenie zvukových dát.

```
//volanie triedy pre vytvorenie zvukových dát
new SynGen().getSynthesizData(audioData, CalculateRGB)
```

Veľkosťou poľa vytvoreného pre spätné vloženie audio dát (premenná *audioData*) určujeme taktiež dĺžku výsledného samplu. Dĺžka je tiež závislá od počtu kanálov prehrávania – mono/stereo. V premennej *CalculateRGB* sa nachádzajú dáta potrebné k vytvoreniu zvukovej vzorky.

V triede *SynGen* sú zadeklarované premenné, ktoré slúžia pre konvertovanie a uloženie dát, z ktorých možno vytvoriť prekladateľné alebo ukladateľné médium. Funkciou *getSynthesizData* riadime proces, v ktorom sa určuje, aký typ dát sa bude vytvárať, podľa voľby užívateľa. Funkciou je možné vytvoriť nespracovaný signál alebo efektovo spracovaný signál, taktiež sa v nej nachádzajú funkcie volajúce proces filtrácie.

Dáta bez efektových procesov sú vytvárané funkciou *tones*, zabezpečujúcou generovanie signálu v závislosti od parametrov veľkosti poľa, do ktorej sú dáta vložené, voľby mono kanálu, samplovacej dĺžky a samotného vloženia každej časti signálu generovaného funkciou *getFinalData* do poľa pre prehrávanie alebo ukladanie zvuku.

Kódové riešenie na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.7: Trieda SynGen.

```
void getSynthesizData(byte[] synDataBuffer, int [] data){
    //premenne pre ukladanie zvukovych dat
    byteBuffer = ByteBuffer.wrap(synDataBuffer);
    shortBuffer = byteBuffer.asShortBuffer();
    byteLength = synDataBuffer.length;
    tones(data);          //funkcia pre vytvorenie dat}
void tones(int [] data){
    int bytesPerSamp = 2;
    channels = 1;           //mono kanal
    int sampLength = byteLength/bytesPerSamp;
    for(int cnt = 0; cnt < sampLength; cnt++){
        double time = cnt/sampleRate;
        double finalData = getFinalData(data, time);
        shortBuffer.put((short)( 16384 * finalData));
    }
}
```

4.5.1 Generovanie syntetického zvuku

Vo funkcii *getFinalData* vytvárame syntetický zvukový materiál matematickou funkciou sínus, ktorej hodnoty pre vytvorenie frekvencie a amplitúdy dostávame z informácií v predošlom spracovaní popísaného v kapitole 4.4. Činnosť funkcie je založená na teórii o aditívnej syntéze, uvedenej v kapitole 2.4.1. Funkcia generuje jednotlivé časti digitálneho signálu v závislosti od vstupných dát a času podľa matematického vzorca (2.1). Taktiež zabezpečuje vytvorenie oktáv v závislosti od výskytu bielej alebo čiernej farby, podrobnejšie vysvetlené v kapitolách 4.4.1 a 4.4.2.

K jednotlivým číselným označeniam farieb sú priradené frekvencie podľa obr. 4.2. Taktiež sú vypočítané percentuálne hodnoty amplitúd v závislosti od výskytu jednotlivých farieb. Výpočet najprv zistí najviac obsiahnutú farbu, z ktorej berie hodnotu maximálnej amplitúdy, ostatné amplitúdy sú tvorené podielom zo zisteného maxima. Programové riešenie je rozdelené na dve základné časti, prvá časť rieši vytvorenie zvukovej ukážky z čierno-bielej vzorky, druhá časť sa podieľa na ostatných farebných hodnotách preložených do zvukovo spracovateľného formátu.

Kódové riešenie na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.8: Generovanie syntetického zvuku

```
private double getFinalData(int [] data, double time){
    double output = 0; //premenna pre vystup dat
    int[] freq = new int[] {440, 0, 349, 370, 392, 415, 440, 466, 494, 523, 554, 587,
    622, 659}; //jednotlive frekvencie farieb
    boolean UP = false, DOWN = false;
    double max = 0, ampl = 0;
    //zistenie maxima
    for(int i = 0; i < 14; i++) {
        if (max <= data[i]) {
            max = data[i];}
    //riesenie oktavy
    if((data[0]/max)*100 > 50){
            DOWN = true;
    }else if((data[1]/max)*100 > 50){
            UP = true;}
  //riesenie ciernobiela
  var sin = Math.sin(2 * Math.PI * freq[0] * time);
  if(UP){
    output = (sin + (0.5 * Math.sin(2*Math.PI*(freq[0]*2)*time)));
  }else if(DOWN){
    output = (sin + (0.5 * Math.sin(2*Math.PI*((freq[0])/2.0)*time)));
  }else{
    output = sin;}
  //ostatne riesenie farieb
  for(int i = 2; i < 14; i++){
    ampl = (data[i]/max);
    output = output + (ampl * (Math.sin(2 * Math.PI * freq[i] * time)));
    if(UP){
        output = output + (0.5 * ampl * (Math.sin(2 * Math.PI * (2 * freq[i]) * time)));
    }else if(DOWN){
      output = output + (0.5 * ampl * (Math.sin(2 * Math.PI * (0.5 * freq[i]) * time)));
    }}
return output;
}
```

4.6 Signálové spracovanie

Vytvorený nástroj ponúka viacero možností pre úpravy a spracovania výsledného zvuku. V kapitole 4.6 budú popísané možnosti ponúkané vytvoreným nástrojom, ich rozdelenie, funkčnosť a kódové riešenie.

Práca realizuje dve metódy vytvárania signálových spracovaní. Prvá metóda spočíva vo vytvorení úplne nového zvukového materiálu s pridaným procesingom spracovania a druhá metóda vychádza zo spracovania vytvoreného signálu vybraným typom procesingu. Podľa grafického návrhu na obr. 4.1 možno rozdeliť spracovania do skupiny efekty, filtrácia a registre. Nastavovanie módu (tlačidlo MODE) prehrávania je realizované taktiež určitým typom filtrácie – zaradené do skupiny filtrácia.

4.6.1 Efektové spracovanie

Efektové procesy sú realizované metódou vytvárania nového zvukového materiálu s pridaným procesingom efektu podľa voľby užívateľa. Na výber sú spolu štyri typy efektov: PingPong, Decay, Echo a Stereo Panning efekt. Po stlačení jednotlivého efektu je volaná funkcia *pressEffect*, v ktorej sú všetky efekty vložené do jednej skupiny a vytváranie nového signálu so zvoleným efektovým spracovaním zabezpečuje trieda *SynGen* s funkciou *getSynthesizData* (viac v kapitole 4.5), v ktorej je podmienkovou štruktúrou zvolený a vytvorený nový efektový zvukový formát dát.

Kódové riešenie na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.9: Volanie efektového spracovania.

```
public void pressEffect(ActionEvent event){
    ToggleGroup group = new ToggleGroup(); //Skupina efektov do ktorej su vlozene jednotlive efekty
    PingPong.setToggleGroup(group);
    Decay.setToggleGroup(group);
    Echo.setToggleGroup(group);
    Panning.setToggleGroup(group);
    NoEffect.setToggleGroup(group); //tlacidlo bez efektoveho spracovania
    new SynGen().getSynthesizData(audioData, CalculateRGB);
}

void getSynthesizData(byte[] synDataBuffer, int [] data){
    byteBuffer = ByteBuffer.wrap(synDataBuffer);
    shortBuffer = byteBuffer.asShortBuffer();
    byteLength = synDataBuffer.length;
    \//ednotlive efekty
    if(PingPong.isSelected()){pingPong(data);}else if(Decay.isSelected()){decay(data);
    }else if(Echo.isSelected()){echo(data);}else if(Panning.isSelected()){panning(data);
    }else{tones(data);\\bez efektoveho spracovania}
    ...atd...
```

Pre každý typ spracovania je vytvorená vlastná funkcia, označená menom efektu, v ktorej vzniká proces samotného vytvorenia. Funkciou *getSynthesizData* možno vybrať z piatich zvukových foriem, z ktorých sú štyri efektovo spracované a jeden formát zvuku bez efektového spracovania volaný funkciou *tones*.

Efekty sú vytvárané alebo na základe časového spracovania signálu vrátane opakovania či posunu, alebo využívajú vlastnosti stereofónneho posluchu a signál smerujú medzi pravým a ľavým reproduktorom. Taktiež pracujú s jednotlivým zosilnením alebo zoslabením (gain) signálu. Zosilnenia sú realizované pri mono dátach násobiteľom 16384, čo je číslo z polovice maximálnej hodnoty dátového typu *short* a násobiteľom 8192 pri stereofónnom posluchu, čo je polovica hodnoty z mono kanálu.

Stereo Panning efekt

Efekt využíva vlastnosť stereo reproduktorov, vstupujúci signál smeruje do ľavého a pravého reproduktoru striedavo. Amplitúda signálu sa mení z ľavého reproduktoru do pravého v závislosti na čase. V prípade nášho softvéru čas menenia závisí od samplovacej frekvencie. Pri stereofónnom signále treba vytvárať zvukovú vzorku so štyrmi bytmi na jeden sampel. Riadenie procesu pre vytvorenie efektu je realizované cyklom for.

Kódové riešenie na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.10: Efekt Stereo Panning.

```
void panning(int [] data){
    channels = 2;//Stereo kanal
    int bytesPerSamp = 4;//V zavislosti od kanalov
    int sampLength = byteLength/bytesPerSamp;
    for(int cnt = 0; cnt < sampLength; cnt++){
        //Vypocet gainu pre kazdy kanal v zavislosti od casu
        double rightGain = 8192 * cnt/sampLength;
        double leftGain = 8192 - rightGain;
        double time = cnt/sampleRate;
        //Data pre lavy kanal
        double finalData = getFinalData(data, time);
        shortBuffer.put((short)(leftGain*finalData));
        //Data pre pravy kanal
        shortBuffer.put((short)(rightGain*finalData));
    }//koniec cyklu
}//koniec metody panning
```

Stereo PingPong efekt

Podobne ako u efektu Stereo Panning využíva tento efekt stereofónneho posluchu, na rozdiel od predchádzajúceho efektu, sa amplitúda nemení postupne, ale signál mení gain rázovo medzi pravým a ľavým kanálom v závislosti od polovice dĺžky samplu. Efekt nemá plynulý prechod signálu medzi kanálmi, ale okamžitý. Riadenie je realizované znova for cyklom.

Kódové riešenie na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.11: Jadro efektu Stereo PingPong.

```
...
double leftGain = 0, rightGain = 8192;
for(int cnt = 0; cnt < sampLength; cnt++){
    if(cnt % (sampLength/2) == 0){
        //razova vymena gainu
        double temp = leftGain;
        leftGain = rightGain;
        rightGain = temp;
    }...
```

Decay efekt

Mono efekt spôsobuje klesanie amplitúdy v závislosti na čase lineárnym priebehom. Efekt signál postupne stišuje. Klesanie je závislé od hodnoty premennej *scale*, bližšie vysvetlené v kódovej ukážke.

Kódové riešenie na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.12: Efekt Decay.

```
void decay(int [] data){
    channels = 1;//Mono efekt
    int bytesPerSamp = 2;//V zavisloti od kanalov
    int sampLength = byteLength/bytesPerSamp;
    for(int cnt = 0; cnt < sampLength; cnt++){
//Hodnota scale ovplyvnuje rychlost klesania - vacsia hodnota scale,rychlejsie klesanie.
        double scale = 2*cnt;
        if(scale > sampLength) scale = sampLength;
        double gain = 16384*(sampLength-scale)/sampLength;
        double time = cnt/sampleRate;
        double finalData =getFinalData(data, time);
        shortBuffer.put((short)(gain*finalData));
    } }//koniec metody decay
```

Echo efekt

Efekt echo spôsobuje reflexie vstupujúceho signálu, ktoré sú pravidelne časovo posunuté s postupne klesajúcou amplitúdou. K zjednodušeniu spojenia jednotlivých reflexií signálu je využitá pomocná funkcia *echoPulseHelper*. Podrobnejšie vysvetlené v kódovej ukážke.

Kódové riešenie na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.13: Efekt Echo.

```
void echo(int [] data){
    channels = 1;//Mono posluch
    int bytesPerSamp = 2;//Zavisle na kanaloch Mono/Stereo
    int sampLength = byteLength/bytesPerSamp;
    int cnt2 = -10000;//casovy posun prvej reflexie
    int cnt3 = -20000;//casovy posun druhej reflexie
    int cnt4 = -30000;//casovy posun tretej reflexie
    for(int cnt1 = 0; cnt1 < sampLength; cnt1++,cnt2++,cnt3++,cnt4++){
        double val = echoPulseHelper(cnt1,sampLength, data); //zakladny signal do ktoreho su pridane
            reflexie
        if(cnt2 > 0){//prva reflexia s nizsim zosilenim (gain = 0,7)
            val += 0.7 * echoPulseHelper(cnt2,sampLength, data); }
        if(cnt3 > 0){//druha reflexia s nizsim zosilenim (gain = 0,49)
            val += 0.49 * echoPulseHelper(cnt3,sampLength, data); }
        if(cnt4 > 0){//tretia reflexia s nizsim zosilenim (gain = 0,34)
            val += 0.34 * echoPulseHelper(cnt4,sampLength, data); }
        shortBuffer.put((short)val); } }
double echoPulseHelper(int cnt,int sampLength, int[] data){
    double scale = 2*cnt;
    if(scale > sampLength) scale = sampLength;
    double gain = 16384*(sampLength-scale)/sampLength; double time = cnt/sampleRate;
    double finalData = getFinalData(data, time);
    return(short)(gain*finalData);}
```

4.6.2 Filtračné spracovanie

Filtračnými procesmi realizujeme v našej práci možnosť úpravy vytvorených dát. Filtrácia vychádza z teoretických poznatkov uvedených v kapitole 2.5. Vygenerované dáta použiteľné pre vytvorenie prehrávateľného alebo uložiteľného audio súboru spracovávame triedou *Filter*, v ktorej je možné realizovať štyri typy filtrov: hornú priepusť, dolnú priepusť, pásmovú priepusť a pásmovú zádrž. Filtre sú programovo pevne nastavené na jednotlivé frekvenčné hodnoty.

Hlavným dôvodom vloženia možnosti filtrácie do našej práce bolo odstránenie postranných frekvenčných pásiem vzniknutých pri generovaní audio dát do určitého časového okna. Odstránenie postranných pásiem bolo realizované filtrom typu pásmová priepusť, ktorého sme stredný kmitočet nastavili na hodnotu 550Hz. Hodnota vychádza z približného logaritmického stredu nástrojom generovaných kmitočtov, ktoré sa pohybujú v rozmedzí zaokrúhlene od 100Hz do 2500Hz. Pre docielenie strmšej charakteristiky sme zvolili kvality koeficient Q na hodnotu 3, tým sme docielili pokles na krajných hodnotách nástroja o približne 25 decibelov. Možnosťou filtrácie na určitom frekvenčnom spektre pomocou pásmovej priepusti realizujeme funkciu *BaseBand* v časti nástroja pomenovanej MODE, ktorá je znázornená v grafickom návrhu na obr. 4.1. Inverzným riešením je realizovaná funkcia *SideBand*, kedy využívame rovnakej frekvenčnej hodnoty 550Hz, ale s typom filtru notch, respektíve pásmová zádrž. V tomto riešení sme zvolili pre docielenie strmšej frekvenčnej charakteristiky hodnotu kvality koeficientu Q na číslo 0,05. Pri krajných frekvenciách sme sa týmto nastavením dostali na hodnotu poklesu približne 10 decibel. Dôvodom realizácie funkcie bolo využiť vzniknuté postranné pásma pre ich experimentálne vlastnosti a možnosti použitia. Posledným použitím filtrácie sme realizovali pevne nastavené filtre typu horná a dolná priepusť, ktoré sa nachádzajú v časti nástroja Filter znázornenom na obr. 4.1. Filter hornej priepusti pod názvom na grafickom návrhu LoCut má nastavenú krajnú frekvenciu na hodnotu 100Hz. Hodnota krajného kmitočtu dolnej priepusti označenej ako HiCut je nastavená na 2,5kHz. Obe hodnoty vychádzajú z šírky generovaného pásma. Dôvodom vloženia pevne nastavených filtrov je využitie triedy *Filter*, realizovanej pre filtrácie postranných pásiem a pridanie ďalšej možnosti užívateľovi pre vytvorenie alebo prípadné porovnanie vytvorených zvukových ukážok. Po stlačení voľby filtrácie užívateľom (MODE alebo LoCut, HiCut) je volaná funkcia *pressRegenerate*, v ktorej sa vytvorí skupina tlačidiel MODE a volanie triedy *SynGen*, ktorá po vygenerovaní obsahuje taktiež odkaz na jednotlivé filtračné procesy zvolené užívateľom nachádzajúce sa v triede *Filter*. Trieda *Filter* pracuje na základe teórie o bikvadratických filtroch, bližšie v kapitole 2.5.4. Jednotlivé členy vytvorených zvukových dát sú posielané do triedy *Filter* a na základe teórie o filtrácii prepisované podľa typu zvoleného filtra.

Kódové riešenie na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.14: Funkcia pressRegenerate.

```
public void pressRegenerate(ActionEvent event){
    ToggleGroup group2 = new ToggleGroup();
    NormMode.setToggleGroup(group2);
    BaseMode.setToggleGroup(group2);
    SideMode.setToggleGroup(group2);
    new SynGen().getSynthesizData(audioData, CalculateRGB);
}
```

Filtračné procesy sú v triede *SynGen* zaradené za efektovým spracovaním jednotlivých zvukových dát. Riadenie jednotlivých filtrácií je zabezpečované podmienkovými štruktúrami, v ktorých sú informácie konvertované na dátový typ *float*, vytvoria sa filtračné koeficienty zo zvolených parametrov filtra a dáta sú posielané funkciou *Update* a prepisované na základe ich vstupných informácií a jednotlivých vypočítaných koeficientov pomocou funkcie *getValue*, nachádzajúcej sa v triede *Filter*.

Kódové riešenie na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.15: Volanie filtrácie v triede SynGen.

```
...trieda SynGen po efektovom spracovani...

if(SideMode.isSelected()){
    float[] array = new float[audioData.length/2]; //vytvorenie premennej float
    //konvertovanie funkciou pre 16 bit, PCM signed, BigEndian, typ slova float
    new AudioFloatConversion16SB().toFloatArray(audioData,0, array,0,array.length);
    //volanie triedy Filter pre vytvorenie koeficientov zo zadanych parametrov
    Filter filter = new Filter(510, 44100, Filter.PassType.Notch);

    //hlavny proces filtracie kazdeho vzorku
    for(int a = 0; a < array.length; a++){
        filter.Update(array[a]);
        array[a] = filter.getValue();
}
//konvertovanie nazad funkciou pre 16 bit, PCM signed, BigEndian, typ slova byte
new AudioFloatConversion16SB().toByteArray(array,0,array.length, audioData, 0);}

//dalsia mozna filtracia podla volby uzivatela
else if(BaseMode.isSelected()){
    float[] array = new float[audioData.length/2];
    new AudioFloatConversion16SB().toFloatArray(audioData,0, array,0,array.length);
    Filter filter = new Filter(510, 44100, Filter.PassType.Bandpass)
```

Výpis kódu 4.15 zobrazuje spôsob, ktorým realizujeme v našej práci všetky typy filtrácie. Riadenie filtračného procesu zabezpečuje cyklus for, v ktorom sú jednotlivé vzorky dát prepisované novými filtrovanými vzorkami vytvorenými triedou *Filter*.

Samotný filtračný proces sa nachádza v triede *Filter*, v ktorej sú vytvorené koeficienty na základe typu filtra a podľa rovnice (2.9) prepísanej do programovej logiky sú vytvárané výstupné dáta.

Kódové riešenie na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.16: Koeficienty triedy Filter.

```
public static class Filter {
   private double a1, a2, a3, b1, b2; //premenne koeficientov filtru
   private float[] inputHistory = new float[2];//premenna historie vstupnych dat
   private float[] outputHistory = new float [2];//premenna historie vystupnych dat
   ... pokracovanie triedy Filter ...
```

Jednotlivé koeficienty sú vytvárané funkciou *Filter*, výpočet vychádza z matematických vzorcov prenosových funkcií pre určité typy filtrov uvedených podrobnejšie v kapitole 2.5.3. Pri filtroch typu horná a dolná priepusť realizujeme koeficient kvality Q hodnotou 0,707 vychádzajúci z Butterworthovej aproximácie analógových filtrov.

Kódové riešenie na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.17: Funkcia Filter.

```
Filter(int frequency, int sampleRate, @NotNull PassType passType){
       double c = Math.tan(Math.PI * frequency / sampleRate);//prewarp
       double norm;
       switch (passType) {
       //matematicky zjednodusene vypocety jednotlivych koeficientov typu filtra
           case Lowpass:
               norm = 1 / (1 + c / 0.707 + c * c);
               a1 = c * c * norm;
               a2 = 2 * a1;
               a3 = a1;
               b1 = 2 * (c * c - 1) * norm;
               b2 = (1 - c / 0.707 + c * c) * norm;
               break;
           case Highpass:
               norm = 1 / (1 + c / 0.707 + c * c);
               a1 = 1 * norm;
               ...
```

Programová realizácia funkcie pre vytvorenie filtrovaného výstupu je pomocou *Update* funkcie a vrátenie pomocou funkcie *getValue* (výpis kódu 4.18).

Kódové riešenie na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.18: Filtrovaný výstup.

```
void Update(float newInput) {
    //vytvorenie vystupu
    float newOutput = (float) (a1 * newInput + a2 * this.inputHistory[0] +
    a3 *   this.inputHistory[1] - b1 * this.outputHistory[0] - b2 * this.outputHistory[1]);
    //riesenie rekurzie z predchadzajucich hodnot
    this.inputHistory[1] = this.inputHistory[0];
    this.inputHistory[0] = newInput;
    this.outputHistory[1] = this.outputHistory[0];
    this.outputHistory[0] = newOutput; }
float getValue() {
    return this.outputHistory[0];
}}
..koniec triedy Filter
```

Kódové riešenia popisujú realizáciu digitálnej filtrácie poznatkami z teoretických informácií digitálnej filtrácie. Vytvorením digitálneho lineárneho bikvadratického filtra umožňujeme užívateľovi realizáciu zvukových dát bez vzniknutých postranných pásiem alebo zvukových dát obsahujúcich len vzniknuté postranné pásma. Taktiež poskytujeme použitie pevne nastavených filtrov typu horná a dolná priepusť pre vytvorenie alebo porovnanie vytvorených audio ukážok softvérovým nástrojom.

4.6.3 Vytvorenie registrov

Posledným spôsobom úpravy dát poskytujeme užívateľovi pridanie registrov z dôvodu vytvorenia širších spektrálnych vlastností experimentálneho nástroja. Jedná sa o pridanie frekvenčných hodnôt vychádzajúcich z vloženého obrázkového súboru, ktorých vlastnosti frekvencie budú závislé na vloženom obrázkovom médiu. Realizujeme možnosti vytvorenia frekvencií s oktávovým alebo kvintovým posunom frekvenčne vyšších alebo nižších vlastností.

Oktávovým posunom chápeme násobok alebo podiel (závisí od voľby vzniku vyššej alebo nižšej frekvencie) frekvenčnej hodnoty vzniknutej z vloženého obrázkového média číslom 2. Kvintovým posunom chápeme násobok alebo podiel frekvenčnej hodnoty z vloženého média zlomkom 3/2 pre vznik vyšších alebo nižších frekvenčných parametrov. Keďže vznik oktávového posunu taktiež realizujeme z vlastností bielej alebo čiernej farby z vloženého média (podrobnejšie v kapitole 4.4), užívateľovi poskytujeme možnosť vytvorenia registru o dve oktávy vyššie alebo nižšie, aby sa predchádzalo k možnému vzniku kauzy vytvorenia dvoch rovnakých frekvencií. Voľba pridania registrov je znázornená na obr. 4.1 pod skupinou Registers.

Spolu možno realizovať štyri rozšírenia spektrálnych vlastností nástroja obsahujúcich register o dve oktávy frekvenčne vyšší, o dve oktávy frekvenčne nižší, o kvintu vyšší a o kvintu frekvenčne nižších vlastností.

Po stlačení jednotlivých registrov je v programovom riešení volaná funkcia *pressRegenerate*, ktorú zobrazuje výpis kódu 4.14, ktorá realizuje pridanie registrov pri vytváraní zvukových dát, nezávisle na zvolenom efekte vo funkcii *getFinalData*, ktorú popisuje výpis kódu 4.8. Do funkcie *getFinalData* boli pridané podmienkovou štruktúrou jednotlivé možnosti pridania registrov.

Kódové riešenie naväzuje na výpis kódu 4.8 a je naprogramované autorom práce.

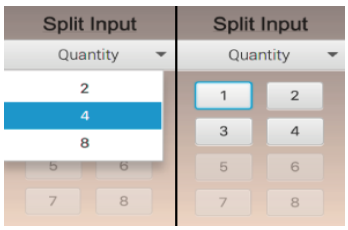
Výpis 4.19: Pridanie registrov.

```
...pokračovanie funkcie getFinalData...
    }else if(DOWN){
     output = output + (0.5 * ampl * (Math.sin(2 * Math.PI * (0.5 * freq[i]) * time)));
    }//register dve oktavy vyssie
    if(OctUp.isSelected()){
        output = output + (0.3 * ampl * (Math.sin(2 * Math.PI * (4 * freq[i]) * time)));
    }//register dve oktavy nizsie
    if(OctDown.isSelected()){
        output = output + (0.8 * ampl * (Math.sin(2 * Math.PI * (freq[i]/4.0) * time)));
    }//register kvinty vyssie
    if(FifthUp.isSelected()){
      output = output + (0.3 * ampl * (Math.sin(2 * Math.PI * (1.5 * freq[i]) * time)));
    }//register kvinty nizsie
    if(FifthDown.isSelected()){
      output = output + (0.5 * ampl * (Math.sin(2 * Math.PI * (3*freq[i]/4.0) * time))); }
return output;
}//koniec funkcie getFinalData vratane moznosti registrov
```

Amplitúdy generovaných signálov s vyššie položenými frekvenciami boli zvolené na hodnotu 30 percent. Pri realizácii registrov s nižším kmitočtom sme amplitúdy volili pre lepšiu počuteľnosť s hodnotami 50 a 80 percent.

4.7 Rozdelenie obrázkového média

Užívateľovi je umožnené rozdeliť vstupný obrázkový materiál na dva, štyri či osem rovnakých segmentov, ktoré je možné osobitne spracovávať jednotlivo, nezávisle od seba. Každé nastavenie je v zariadení ukladané a vo výsledku je možné spätne upravovať segmenty viackrát. Realizácia rozdelenia vstupného obrázkového média je tvorená rozpočítaním vloženého súboru na rovnaké časti podľa počtu zvoleného užívateľom zo zistených parametrov obrázka, jeho šírky a výšky. Možnosť rozdelenia a prepínanie medzi segmentami je realizované tlačidlami znázornenými na obr. 4.5, kde pod tlačidlom *Quantity* volíme počet častí a jednotlivými číselnými označeniami je realizovaný výber upravovaného segmentu.



Obr. 4.5: Rozdelenie obrázkového materiálu.

Po výbere počtu, do ktorého vložené obrázkové médium bude rozdelené, sú volané jednotlivé funkcie pre rozdelenie do dvoch, štyroch alebo ôsmich častí, ktoré sú pomenované *pressSplit2, pressSplit4, pressSplit8*. Princípy funkcií fungujú rovnakým spôsobom, s rozdielom počtu vytvorených nových obrázkových segmentov. Po rozdelení je prvotne nastavené pracovanie so segmentom číslo 1. Rozloženie segmentov zobrazené na obr. 4.5 zodpovedá aj reálnemu rozloženiu rozdelenia vloženého obrázkového média. Pre ukladanie nastavenia jednotlivých parametrov pre každý segment samostatne je vytvorená pomocná premenná poľa *history*, do ktorej sa pri jednotlivých spracovaniach ukladajú nastavenia. Práca s jednotlivými segmentmi je rovnaká ako práca s vloženým obrázkovým materiálom. Užívateľ má možnosť využiť efektové aj filtračné spracovanie, prípadné zapnutie registrov, alebo meniť možnosti mapovania do akronym farieb HSV alebo RGB. Po stlačení výberu rozdelenia sa nastavenia uložené v histórii automaticky vymazávajú. Výpočet rozdelenia vychádza z pravidelného delenia vloženého obrázka na rovnaké časti, v prípade, že sa vložený obrázok skladá z nepárneho počtu pixelov, je kódové riešenie ošetrené a vytvára rozdelenie do častí, z ktorých je jedna o nepárny pixel rozšírená. Príklad rozdelenia do dvoch častí, po ktorom je volaná funkcia *pressSplit2* zobrazuje výpis kódu 4.20.

Kódové riešenie na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.20: Rozdelenie obrázka na dve časti.

```
public void pressSplit2(ActionEvent event)throws IOException{
        BufferedImage imageReal = ImageIO.read(new File(path));
        int w = imageReal.getWidth();//zistenie sirky obrazkoveho media
        int h = imageReal.getHeight();//zistenie vysky obrazku
        int mid = w/2;//vypocet stredu
        int mid2 = mid;
        //vypocet stredu pri neparnej sirke obrazkoveho media
        if (mid % 2 != 0){
            mid = mid+1;
            mid2 = mid-1; }
        //vytvorenie dvoch segmentov obrazkoveho suboru z vypocitaneho stredu
        img[1] = imageReal.getSubimage(0, 0, mid, h);
        img[2] = imageReal.getSubimage(mid, 0, mid2, h);
        //defaultne spracovanie obrazkoveho suboru z prveho segmentu
        CalculateRGB = convert(img[1],hsv);
        BtnNum = 1;
        //vycistenie historie nastavenia
        for(int a = 0; a < 9; a++){
            for(int b = 0; b < 4; b++){
                history[a][b] = 0;
            } }
        //vytvorenie prehravatelneho typu audio dat
        new SynGen().getSynthesizData(audioData, CalculateRGB);
        //povelenie prvych dvoch tlacidiel pre rozdelenie obrazku
        btn1.setDisable(false);btn2.setDisable(false);
        for(int i = 3; i<9; i++){
            btn[i].setDisable(true); }
        //oznacit tlacidlo 1 ktore sa momentalne pouziva
        btn1.requestFocus(); }
```

Po každom stlačení číselného označenia jednotlivej časti segmentu je volaná funkcia *pressNumSplit* slúžiaca ku generovaniu zvukových dát aktuálne stlačeného segmentu. Zistenie stlačeného segmentu je realizované triedou *GetBtnNum*.

Kódové riešenie na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.21: Generovanie zvuku zo zvoleného segmentu.

```
public void pressNumSplit(ActionEvent event){
    for(int i = 1; i< 9; i++){
//zistenie ktore tlaacidlo bolo stlacene
        btn[i].addEventHandler(MouseEvent.MOUSE_CLICKED, new GetBtnNum());
    }
//generovanie prislusnych zvukovych dat
    CalculateRGB = convert(img[BtnNum],hsv);
    new SynGen().getHistSynthesizData(audioData, CalculateRGB);
    btn[BtnNum].requestFocus();}

class GetBtnNum implements EventHandler<Event>{
    @Override
    public void handle(@NotNull Event event){
//ziskanie cisla tlacidla z JavaAPI
        String id = event.getSource().toString();
        String []a = id.split("n");
        String []out = a[2].split(",");
        String fnl = out[0];
        BtnNum = Integer.parseInt(fnl);
}}
```

Rozdelením vloženého obrázkového média možno docieliť lepšie pochopenie fungovania nástroja a zároveň možno vytvoriť alebo porovnať vytvorené zvukové dáta.

4.8 Prehrávanie a ukladanie zvukových dát

Kapitola 4.8 je zameraná na vysvetlenie problematiky zaoberajúcej sa vytvorením prehrávateľného alebo uložiteľného formátu zvukových dát v programovom riešení. Teoretické poznatky vytvárania zvukových ukážok vychádzajú z kapitoly 2.7. Nasledujúce programové riešenia nadväzujú na predom pripravený obsah dát, ktorý je možno dostať krokmi popísanými v kapitole 4.5.

Vytváraný formát sa skladá z pevne stanovenej štruktúry, popísanej v kapitole 2.7.2, do ktorej sú vygenerované dáta vložené. Vo formáte sa zadáva samplovacia frekvencia, ktorá realizuje diskrétny časový signál, počet kanálov, ktoré realizujú prehrávanie typu mono, čo je jeden kanál a signál sa posiela do všetkých reproduktorov rovnako, alebo typu stereo, kedy sú kanály dva nezávisle riešené a typ zoradenia dát big-endian.

Premenná typu *AudioInputStream* zabezpečuje vytvorenie vlákna s vloženými dátami v bajtovom poli typu *ByteArrayInputStream* a vytvoreným formátom.

Kódové riešenie na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.22: Vytvorenie audio formátu a zvukových dát.

```
//vlozenie bytoveho pola s vygenerovanymi datami do premennej typu InputStream
InputStream byteArrayInputStream = new ByteArrayInputStream(audioData);

int sampleSizeInBits = 16;//pocet bitov na sampel

encoding = AudioFormat.Encoding.PCM_SIGNED;//pulzna kodova modulacia

//vytvorenie audio formatu
audioFormat = new AudioFormat(
        encoding,
        sampleRate,          //samplovacia frekvencia
        sampleSizeInBits,    //velkost samplu v bitoch
        channels,            //počet kanálov MONO/STEREO 1/2
        ((sampleSizeInBits/8)*channels),
        sampleRate,
        BigEndian);

//vytvorenie vlakna do ktoreho su vlozene data s prislusnym formatom
audioInputStream = new AudioInputStream(byteArrayInputStream, audioFormat,
     audioData.length/audioFormat.getFrameSize());
```

4.8.1 Prehrávanie spracovaných zvukových dát

Prehrávanie je realizované funkciou *pressPlay*, ktorá je volaná po stisnutí tlačidla Play na obr. 4.1. Funkcia najprv vytvorí zvukový formát dát, ktorý zobrazuje výpis kódu 4.22, dátovú linku pomocou triedy *DataLine* nachádzajúcej sa v JavaAPI a prehrávanie realizované triedou *ListenThread*.

Kódové riešenie na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.23: Vytvorenie prehrávacej linky.

```
public void pressPlay(ActionEvent event){

... vytvorenie audio formatu je zobrazene na vypise kodu 4.22 ...

//vlozenie informacii o formate do triedy zabezpecujucej vytvorenie datovej linky
DataLine.Info dataLineInfo = new DataLine.Info(SourceDataLine.class, audioFormat);

sourceDataLine = (SourceDataLine) AudioSystem.getLine(dataLineInfo);

//trieda spustajuca a riadiaca prehravanie
new ListenThread().start();}//koniec funkcie pressPlay
```

Riadiaci proces prehrávania v triede *ListenThread* pomocou dátovej linky typu *SourceDataLine*, ktorú obsahuje JavaAPI. Vytvorenie priebehu prehrávania je zobrazené na nasledujúcej programovej ukážke.

Kódové riešenie na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.24: Prehrávanie zvukového formátu dát.

```
class ListenThread extends Thread{
   byte[] playBuffer = new byte[audioData.length];//premenna o velkosti prehravanych dat
    public void run(){ try{
            sourceDataLine.flush();    //vyprazdnenie predchadzajucich udajov
            sourceDataLine.open(audioFormat); //ziskanie formatu prehravaneho media
            sourceDataLine.start();           //spustenie nadviazania spojenia
            int cnt;
          //prehrávanie
        while((cnt = audioInputStream.read(playBuffer, 0, playBuffer.length)) != -1
                if(cnt > 0){ sourceDataLine.write(playBuffer, 0, cnt); }
        }
        sourceDataLine.drain();             //vyprazdnenie zostavajucich udajov
        sourceDataLine.stop();              //zastavenie spojenia
        sourceDataLine.close();             //ukončenie spojenia
        }catch (Exception e){
            e.printStackTrace();
            System.exit(0); }
     }
}
```

4.8.2 Uloženie spracovaných zvukových dát

Ukladanie dát do počítača je realizované funkciou *pressSave*, ktorá je volaná po stlačení tlačidla Save zobrazenom na obr. 4.1. Funkcia vytvorí rovnaký typ formátu ako pri prehrávaní zobrazuje výpis kódu 4.22. Užívateľ zadá názov, pod ktorým chce dáta uložiť, ktorý je použitý pri ukladaní, v našom prípade je tento názov vložený do premennej *SaveName*. Audio formát, ktorý vytvára softvérový nástroj je typu WAVE (podrobnejšie popísaný v kapitole 2.7.2). Zvukový formát bol vybraný na základe bezstratovej kompresie, jeho všeobecnej použiteľnosti, možnosti konvertovania a predovšetkým uložiteľnosti na prevažnej väčšine počítačových platforiem. Nástroj zvukové médium ukladá do zložky, v ktorej je spustený a po úspešne uložení indikuje textovou správou vloženou spätne do premennej *SaveName*.

Kódové riešenie na nasledujúcej ukážke naprogramované autorom práce.

Výpis 4.25: Ukladanie zvukových dát.

```
public void pressSave(ActionEvent event){
... vytvorenie audio formatu zobrazene na vypise kodu 4.22 ...
    try{
    AudioSystem.write(
            audioInputStream, //premenná v ktorej sú uložené naše audio dáta
            AudioFileFormat.Type.WAVE, //výber formátu akým dáta zapíšeme
            //vytvorenie a uloženie audio média s názvom SaveName
            new File(SaveName.getText() +".wav"));
            SaveName.setText(null);
            SaveName.setPromptText("Saved! Type Next Name");
}catch (Exception e) {
    e.printStackTrace();
    System.exit(0);
}//koniec funkcie pressSave
```

5 Zhodnotenie

Výsledkom realizácie softvérového nástroja je snaha o predvedenie možného prepojenia medzi zvukovými a obrazovými parametrami. Výsledná zvuková ukážka je zložená predovšetkým z informácií nachádzajúcich sa na vloženom obrázkovom médiu a je zároveň ovplyvňovaná kombináciami možností rôznych nastavení ponúkaných nástrojom popísaných v kapitole 4. Praktickým zhodnotením zo strany autora práce zvukového výstupu a celkovej využiteľnosti nástroja sa snažíme priblížiť funkčnosť a zmysel práce.

Zvuková ukážka vytvorená nástrojom pomocou mapovania farebných atribút a prekladu do audio podoby má syntetické vlastnosti, z hľadiska druhu syntézy zvuku prirovnateľná napríklad k nástroju Synclavier II, ktorý má možnosť produkovať zvuky zložené aditívnou syntézou, mimo iných možností. Zo stránky riešenia prekladu dát z obrazu na zvuk je podobný napríklad nástroju Coagula Light, ktorého ale princíp procesu prekladu dát pracuje úplne odlišným spôsobom a nedisponuje napríklad možnosťou rozdelenia vloženého obrázkového média. Zvuk taktiež možno prirovnať štruktúrou na digitálnu 8-bitovú ukážku, so spektrom závislým predovšetkým na vloženom obrázku. Po skúsenosti z viacerých vložených ukážkok sa jedná o zvuk zložený zo 4 až 10 zložiek sínusového signálu (bez pridania alebo volieb možnosti spracovania), ktorý ovplyvňuje spomenuté vložené médium.

Ukážku najviac ovplyvňuje počet vyskytnutých farieb na vloženom obrázkovom súbore, taktiež samotné rozlíšenie obrázka a celková veľkosť. Pri veľkostne priemerných (HD, FullHD) vložených ukážkach je rozhodujúca ich farebná početnosť. Zmenou typu mapovania zvukový výsledok taktiež ovplyvňujeme, keďže každý typ mapovania obsahuje inak nastavené krajné hodnoty jednotlivých farieb. Pri porovnaní sa javí pestrejšie mapovanie do HSV akronym, tvorené zvuky sú spektrálne širšie a frekvenčne vyššie položené oproti mapovaniu typu RGB. Farebné rozmedzia pri HSV sa tiež javia konkrétnejšie, presnejšie ako pri RGB. Tmavosť vloženého obrázka taktiež ovplyvňuje zvukový výsledok – tmavší obrázok generuje nižšie tóny a zvukovú ukážku možno nazvať viac temnú, tajomnú. Toto ovplyvnenie je možné ešte výraznejšie pozorovať po rozdelení vstupného obrázka a prehrávaní jednotlivých segmentov, ktoré sú rôzne farebné. Logicky pri vložení jednofarebného obrázka je zvukový výstup aj po rozdelení do segmentov nezmenený, s chudobným spektrom.

Pridaním efektového spracovania ovplyvňujeme prevažne amplitúdové vlastnosti signálu. Ponúkané stereo efekty sa zdajú viac využiteľné, keďže sa jedná o dvojkanálový posluch. Filtrovanie taktiež signál formuje a možnosť SideBand, kedy sú potlačené vygenerované frekvencie a vystupujú postranné frekvenčné pásma, je zvlášť zaujímavou, vo zvuku možno pri zosilnení vnímať prevažne vysoké frekvencie pripomínajúce kovové alebo sklenené zafarbenie. Problémom je nutnosť veľkého zosil-

nenia signálu. Pri kombinácii efektu Stereo Panning s možnosťou SideBand taktiež dochádza k prechodu medzi ľavým a pravým kanálom, ale s určitým amplitúdovým priebehom, ktorý možno prirovnať napríklad k zvuku morských vĺn.

Výsledná zvuková podoba je sampel, čo značí zvukovú časť zachytávajúcu farbu nástroja. Zvuk je zložený z parametrov vloženého obrázka alebo po rozdelení z vybranej časti obrázka, s možnosťou výberu signálového spracovania. Frekvenčný rozsah nástroja je závislý predovšetkým od farebných vlastností vloženého média, veľkosti a rozlíšenia. Vrátane možností ponúkaných nástrojom zvukový výstup nadobúda frekvenčné hodnoty od 87 Hertz, po približne 2600 Hertz. Počet vygenerovaných sínusoíd signálu sa podľa voľby užívateľa pohybuje od jednej, (jednofarebnéhý obrázok) až po hodnotu dvanástich sínusoíd, ktoré sa môžu reflektovať do piatich oktáv. Spolu nástrojom možno produkovať 60 sínusoíd s jednotlivými frekvenciami a amplitúdami. Amplitúdy frekvencií sú pomerovo rozložené, vzhľadom k maximálnej amplitúde podľa výskytu vo vkladanom obrázkovom médiu. Farba s najväčším výskytom má maximálnu amplitúdu. Dĺžka samplu je závislá od voľby užívateľa, v základnom nastavení je to pre jednokanálový posluch časová hodnota dvoch sekúnd, pre dvojkanálový, resp. STEREO posluch, je to čas jednej sekundy. Výsledný signál je možné uložiť do počítača v bezstratovom zvukovom formáte WAVE.

Pri porovnaní s návrhom uvedeným v semestrálnej práci sa po rozdelení zvuk netvorí spojením všetkých segmentov obrázka, ale každý segment obsahuje svoju vlastnú zvukovú ukážku s možnosťou uloženia a prípadného spracovania. Taktiež pri porovnaní so semestrálnou prácou bola pridaná filtračná časť nástroja, ktorá zvuk ovplyvňuje s hlavným cieľom zamedzenia vplyvu zvukových postranných pásiem vytvorených orezaním generovaných signálov do časového okna a pridaním statických filtrov. Po vytvorení nástroja navrhnutého semestrálnou prácou boli zvukové výsledky pomerne hladké (nepríliš bohaté), do konečného tvaru nástroja bakalárskej práce boli pridané možnosti zväčšenia (rozšírenia) výsledných zvukov pomocou pridania registrov a ďalšieho typu mapovania farebných atribút.

Ako možné využitie vytvorených zvukov nástroja autor pokladá aj prípadné poukázanie možnosti prepojenia obraz – zvuk, ale hlavne použitie vytvoreného samplu ako zvukový doplnok do skladieb elektronických žánrov, kedy z vygenerovaných zvukov možno tvoriť syntetické nástroje, ovládané napríklad pomocou MIDI klaviatúry. Ďalej vrstvením jednotlivých samplov, menením ich dĺžky či inými úpravami možno docieliť ambientných hudobných zvukov, prípadne vychádzajúcich z obrazu konkrétneho prostredia, okolia. Spomenuté sú len možnosti skúsené a aplikované do prác autora, využitie ale silno závisí od kreativity užívateľa. Za isté využitie práce taktiež možno brať popísanú problematiku a jednotlivé kódové riešenia v programovom jazyku Java v prípade s tvorením, spracovaním, prehrávaním a ukladaním zvuku.

6 Záver

Cieľom tejto bakalárskej práce bolo vytvoriť *experimentálny softvérový hudobný nástroj riadený farbou svetla.* Jeho experimentálnosť mala spočívať v jeho netradičnom spôsobe riadenia parametrov zvukovej syntézy pomocou analýzy farieb.

Práca predstavuje možnosť realizácie programového nástroja, ktorý rieši prevod vstupného obrázkového média na zvuk. V prvej časti sa práca sústreďuje na teoretické poznatky z oblasti zvukového a svetelného vlnenia a približuje problematiku farieb. Druhá časť práce rieši vytváranie umelého zvuku pomocou syntéz, z ktorých navrhuje realizáciu pomocou aditívnej syntézy. Poukazuje na problematiku spracovania digitálnych zvukových signálov pomocou filtrácie a príklad aplikácie pri jej využití. Zaoberá sa technikou sonifikácie, ktorú je použitá pri návrhu programového riešenia a oboznamuje s vlastnosťami dátových formátov, z ktorých podrobnejšie popisuje zvolený zvukový formát dát. V návrhovej časti zobrazuje blokovú schému nástroja vrátane funkčného prepojenia. Kapitola 4 - Realizácia popisuje kompletný postup vytvárania softvérového hudobného nástroja, ktorý má experimentálne vlastnosti generovania zvuku pomocou získania farebných atribút. Tvorba zvuku je založená na spájaní jednotlivých spracovaných dát z farieb pomocou aditívnej syntézy. Preklad informácií je realizovaný pomocou programového riešenia založeného na technike sonifikácie v dvoch odlišných typoch mapovania farebných atribút. Sú popísané možnosti spracovania zvukového signálu pomocou jednotlivých efektových programových procesov, pomocou filtračnej triedy a rozšírenie výsledného zvukového spektra pomocou registrov. Práca popisuje rozdelenie vstupného obrázkového média na viaceré časti, ktoré možno spracovávať samostatne pre výsledný experimentálny zvuk. Uvádza kódové riešenia nadviazania spojenia medzi počítačovou zvukovou časťou a zobrazuje vytvorenie štruktúry dát audio formátu pre ukladanie do počítača. V zhodnotení popisujeme výsledný produkt navrhnutý semestrálnou prácou a realizovaný podľa bakalárskej práce. Jeho výsledné vlastnosti, funkčnosť, možnosti a príklady možnej aplikácie.

Systém disponuje grafickým rozhraním, na ktorom je umiestnený vstupný obrázkový súbor a tlačidlami pre ovládanie programu. Grafické rozhranie je realizované pomocou platformy JavaFX a kódové riešenie pomocou jazyku Java bez využitia externých knižníc.

Výsledok realizovaného nástroja má potenciál podnecovať kreativitu pri predstave možného prepojenia medzi obrazom a zvukom, podporovať komponovanie nových zvukov využiteľných, ako možnosť samplu do zvukových skladieb a približovať ukážky využitia procesov spracovania zvukových signálov vrátane ich prejavov.

Literatúra

[1] SVOBODA, Emanuel. *Přehled středoškolské fyziky.* 4., upr. vyd. Praha: Prometheus, 2005. ISBN 978-80-7196-307-3.

[2] PAIN, H. J. *The Physics of Vibrations and Waves.* Sixth Edition. England: John Wiley & Sons, 2005. ISBN 0 470 01296.

[3] CROWELL, Benjamin. *Vibrations and Waves.* California: Light and Matter, 2008. ISBN 0-9704670-3-6.

[4] URBANOVÁ, M.; HOFMANN, J.; ALEXA, P. *Fyzika II: skripta.* Praha: FCHI VŠCHT.

[5] ZWINKELS, Joanne. *Light, Electromagnetic Spectrum.* 2015. 10.1007/978-3-642-27851-8_204-1.

[6] ARNKIL, Harald; ANTER, Karin Fridell; KLARÉN, Ulf. *Colour and Light: Concepts and confusions.* Finland: Aalto University publication series, 2012. ISBN 978-952-60-4609-9.

[7] FAIRCHILD, Mark D. *Color Appearance Models.* Second Edition. England: John Wiley, 2005. ISBN 0-470-01216-1.

[8] ITTEN, Johannes. *The Elements of Color.* USA: Van Nostrand Reinhold Company, 1970. ISBN 0-442-24038-4.

[9] TONNQUIST, Gunnar. *Färgsystemanalys.* Färgantologi bok 3. Stockholm: Byggforskningsrådet, 1995. ISBN 978-91-540-5698-5.

[10] RUSS, Martin. *Sound Synthesis and Sampling.* Third Edition. USA: Focal Press, 2009. ISBN 978-0-240-52105-3.

[11] ZORRILLA, David Martínez. *Synthesizers: A brief Introduction.* North Carolina: Lulu Press, 2008. ISBN 978-1-4092-2251-4.

[12] BEAUCHAMP, James. *Analysis, Synthesis, and Perception of Musical Sounds.* USA: Springer Science + Business Media, 2007. ISBN 978-0387-32496-8.

[13] HERMANN, Thomas. *The Sonification Handbook.* Berlin: Logos Publishing House, 2011. ISBN 978-3-8325-2819-5.

[14] KRAMER, Gregory. *Auditory Display: Sonification, Audification, And Auditory Interfaces.* Santa Fe Institute: CRC Press, 1994. ISBN 978-0201626049.

[15] WORRALL, David. *Sonification Design: From Data to Intelligible Soundfields.* Switzerland: Springer Nature Switzerland, 2019. ISBN 978-3-030-01496-4.

[16] LUCIDCHART. Vytvorené autorom práce pomocou stránky Lucidchart [online]. [cit. 15.11.2019]. Dostupný na: <https://www.lucidchart.com/pages/?noHomepageRedirect=true>.

[17] MELENDEZ, Nicholas. Flutopedia [online]. [cit. 4.11.2019]. Vytvorené na základe dát z URL. Dostupný na: <https://www.flutopedia.com/img/ColorOfSound_Nextdrum_lg.jpg>

[18] RAPIDTABLES. Rapidtables.com. RGB Color Codes Chart. [online]. [cit. 10.12.2019] Dostupný na: <https://www.rapidtables.com/web/color/RGB_Color.html>

[19] BALDWIN, Richard. developer [online]. [cit. 22.11.2019]. Reproduced with permission. Copyright 1999-2019 QuinStreet, Inc. All rights reserved. Dostupný na: <https://www.developer.com/java/other/article.php/2226701/Java-Sound-Creating-Playing-and-Saving-Synthetic-Sounds.html>

[20] FEACHOR, Emmanuel C. a Barrie W. JERVIS. *Digital signal processing: A practical approach.* 2nd ed. Harlow: Prentice Hall, c2002. ISBN 02-015-9619-9.

[21] MNENEY, Stanley H. *An Introduction to Digital Signal Processing: A Focus on Implementation.* Aalborg, DENMARK: River Publishers, 2008. ISBN 978-87-92982-03-2.

[22] SMÉKAL, Z. *Analýza signálů a soustav – BASS.* Brno: FEKT VUT 2012. ISBN 978-80-214-4453-9.

[23] ZUMBAHLEN, Hank, ed. *BASIC LINEAR DESIGN.* Analog Devices, 2007. ISBN 0-916550-28-1.

[24] SMITH III, Julius, Orion. *Introduction To Digital Filters - With Audio Applications.* W3K Publishing, 2007. ISBN 9780974560717.

[25] RUMSEY, Francis. *Desktop Audio Technology: Digital Audio and MIDI Principles.* UK: Focal Press, 2004. ISBN 978-0240519197.