VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

VYTVOŘENÍ PROGRAMŮ PRO PODPORU VÝUKY ZPRACOVÁNÍ SIGNÁLŮ A OBRAZŮ

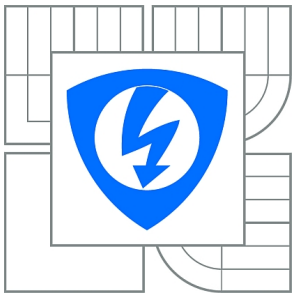
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE LUDĚK NOVOTNÝ
AUTHOR

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

VYTVOŘENÍ PROGRAMŮ PRO PODPORU VÝUKY ZPRACOVÁNÍ SIGNÁLŮ A OBRAZŮ

CREATING PROGRAMS FOR TEACHING SIGNAL AND IMAGE PROCESSING

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

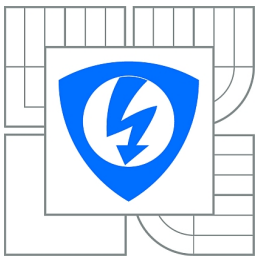
AUTOR PRÁCE — LUDĚK NOVOTNÝ

AUTHOR

VEDOUCÍ PRÁCE — Mgr. PAVEL RAJMIC, Ph.D.

SUPERVISOR

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	Luděk Novotný	***ID:***	146918
Ročník:	3	***Akademický rok:***	2013/2014

NÁZEV TÉMATU:

Vytvoření programů pro podporu výuky zpracování signálů a obrazů

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je vytvořit programy pro interaktivní podporu výuky zejm. v kurzech BZSG, MGMP, BASS. Budou mít podobu JAVA apletů, které se budou tématicky věnovat:1/ názornému vysvětlení fázorů a fázorového diagramu, 2/ filtraci audiosignálu včetně zvukového výstupu, 3/ pseudobarvení šedotónových obrázků, 4/ Huffmanově kódování. Student aplety navrhne, implementuje a ověří v online nasazení.

DOPORUČENÁ LITERATURA:

[1] Beneš, B.; Sochor, J.; Felkel, P.; Žára, J.: Moderní počítačová grafika. Computer Press, Brno, 2005.

[2] Schildt, H. : Java7, výukový kurz, Computer Press, Brno, 2012.

Termín zadání:	10.2.2014	***Termín odevzdání:***	4.6.2014

Vedoucí práce: Mgr. Pavel Rajmic, Ph.D.
Konzultanti bakalářské práce:

doc. Ing. Jiří Mišurec, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Bakalářská práce se zabývá problematikou tvorby interaktivních JAVA apletů. Praktickou část práce tvoří návrh, tvorba a odladění čtyř apletů. První názorně vysvětluje problematiku fázorů, druhý předvádí kmitočtovou filtraci audio signálů, třetí demonstruje umělé barvení černobílých obrázků a čtvrtý vytváří Huffmanův kód ze zadaného textu. Teoretická část se zabývá samotným jazykem JAVA, výhodami apletů a teorií, která se vztahuje k jednotlivým apletům.

KLÍČOVÁ SLOVA

JAVA, aplet, fázor, harmonický signál, zvukový signál, nepravé barvy, Huffmanovo kódování

ABSTRACT

The bachelor thesis deals with the problém of creating interactive JAVA applets. The practical part of work consist of desing, development and debugging four applets. The first one clearly explains the issue of phasor theory, the second one shows frequency filtering of audio signals, the third one demonstrates artificial coloring of grayscale images and the fourth one creates Huffman code from input text. The theoretical part describes the JAVA language, advantages of applets and theory, which relates to particular applets.

KEYWORDS

JAVA, applet, phasor, harmonic signal, audio signal, false color, Huffman coding

NOVOTNÝ, Luděk *Vytvoření programů pro podporu výuky zpracování signálů a obrazů*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2014. 41 s. Vedoucí práce byl Mgr. Pavel Rajmic, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Vytvoření programů pro podporu výuky zpracování signálů a obrazů“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Mgr. Pavlu Rajmicovi, Ph.D. za odborné vedení, konzultace a podnětné návrhy k práci. Děkuji také své rodině, která tolerovala mé dlouhé noční programátorské seance.

Brno ....

(podpis autora)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této bakalářské práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno ....

(podpis autora)

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OBSAH

ÚVOD

Interaktivní forma výuky je v dnešní době velmi žádaná. Pokud student názorně vidí, jak problematika funguje, je schopen ji lépe pochopit a probíranou látku si snáze zapamatovat. Laboratorní úlohy tuto funkci plní velmi dobře. Jsou ale vázány na určitý čas a určité místo. Počítačová simulace může být stejně, ne-li více interaktivní a vzhledem k dostupnosti výpočetní techniky a internetového připojení přístupná takřka kdykoliv. Dobře také doplňuje teoretický výklad, který může být pro studenty často nezáživný.

Tato práce se zabývá návrhem a implementací několika takových interaktivních aplikací, které budou dostupné z internetového prohlížeče. Práce je rozdělena do čtyř částí, přičemž každá se věnuje konkrétnímu apletu a teorii, která se vztahuje k jeho problematice. Spustitelné programy se nacházejí v příloze.

První aplet graficky znázorňuje sčítání fázorů a jejich rotaci. Kromě komplexní roviny dále obsahuje časovou osu, kam se vykresluje časový průběh periodického signálu. Uživatel je schopen vložit vlastní harmonické složky, případně si vybrat některý předpřipravený signál (obdélník, pila, trojúhelník) a zobrazit si jeho ideální průběh, který teoreticky vznikne součtem nekonečného počtu fázorů.

Druhý aplet uživateli demonstruje kmitočtovou filtraci audio signálů. V prvním kroku si uživatel vybere vstupní signál. K dispozici má jak harmonické signály, které aplet za běhu generuje, tak předpřipravené vzorky šumů a krátkých zvukových ukázek. Aplet zobrazuje signál jak v čase, tak jeho frekvenční modulovou charakteristiku. V druhém kroku uživatel vybere ideální filtr a jeho mezní frekvenci, případně střední frekvenci a šířku pásma. Signál se dynamicky filtruje a zobrazuje se filtrovaný signál v čase a jeho frekvenční modulová charakteristika. Důležitou vlastností apletu je, že signál přehrává a uživatel si může poslechnout, jak filtry ovlivňují akustické vlastnosti signálu.

Aplet umělého barvení černobílých obrázků se týká předmětu Základy sazby a grafiky. Vstupní demonstrační obrázek je pixel po pixelu analyzován a šedotónové paletě je přiřazena vybraná barevná. V programu jsou zapracovány palety HSV, Jet, hot a cool. Dále je možné definovat vlastní lineární gradient dvou barev. Uživatel si také může zvolit výstupní gamma funkci.

Poslední aplet vytváří Huffmanův kód na základě vstupního textu. Jednotlivé symboly jsou překopírovány do tabulky, kde dochází ke slučování prvků a přiřazování kódových znaků. Po vytvoření nových znaků se provede výsledné kódování textu.

1 VÝBĚR VHODNÉHO NÁSTROJE

K řešení zadané problematiky se dá přistupovat různými způsoby. Nejprve je potřebné si vysvětlit samotné výhody interaktivních apletů a vybrat vhodný nástroj k jejich tvorbě. V mém případě se jedná o programovací jazyk Java, který si následně lehce popíšeme. Pokusím se též zdůvodnit jeho výběr.

1.1 Aplety a jejich výhody

V dnešní době už není obyklé do počítače instalovat drahé výukové programy z několika desítek CD nosičů. Masové rozšíření internetové sítě a neustálé zvyšování rychlosti připojení umožnilo, aby se tyto aplikace přesunuly do online prostoru a adaptovaly se na toto médium. Pokud jsou demonstrační programy vloženy do webové stránky a dostupné z internetového prohlížeče, nazývají se aplety.

Ty se často různí jak svojí kvalitou, tak svým rozsahem. Bývají to úzce zaměřené interaktivní aplikace pro znázornění omezeného výseku dané problematiky. Díky jejich dostupnosti ale není problém nalézt ten, který uživatel právě potřebuje.

Aplety jsou vázány na různé technologie, což umožňuje vývojářům si zvolit právě tu, se kterou mají nejvíce zkušeností. Mezi ty nejznámější patří Flash od společnosti Adobe nebo programovací jazyk Java od společnosti Sun Microsystems. Jejich společnou vlastností je, že uživatel si musí pro správnou funkci do svého prohlížeče nainstalovat dodatečný software, který se nazývá plugin.

1.1.1 Přístup Javy k řešení bezpečnostních rizik

Pokud se program z prohlížeče sám spouští, přináší to značná bezpečnostní rizika a mohl by tak napáchat v počítači uživatele škodu. Tomu se zamezuje omezenou funkčností apletů, které nemají úplný přístup k systému, na kterém jsou spuštěny.

Bezpečnostní rizika Java řeší velmi elegantně. Výstupem kompilátoru totiž není spustitelný kód, nýbrž bajtkód. Což je sada vysoce optimalizovaných instrukcí, které spouští JVM (*Java Virtual Machine*). Tento interpreter má nad kódem kontrolu a zabraňuje mu vytvářet aktivitu mimo svůj systém. Spouštěný bajtkód v prohlížeči samozřejmě funguje na stejném principu. [1]

1.2 Java

Java je objektově orientovaný programovací jazyk, který byl v roce 1995 vyvinut společností Sun Microsystems, přičemž jeho kořeny prorůstají až do roku 1991, kdy se ještě nazýval Oak. Jeho syntaxe vychází z jazyku C, a má mnoho společných znaků

i s C++. Prvotním impulsem pro jeho stvoření byla potřeba programovacího jazyku, který by bez větších potíží fungoval na široké škále různých druhů CPU (*Central Processing Unit*). To takový C++ umožňoval, nicméně každý procesor vyžadoval vlastní kompilátor, jejichž vývoj byl nákladný. Rozhodně by se tedy takový jazyk nevyplatilo použít pro řízení rozmanitých domácích spotřebičů.

Rozšíření Internetu pak způsobilo, že Java od spotřební elektroniky lehce odstoupila a přeorientovala se na webové aplikace. I zde byla výhodou nezávislost na architektuře. Každý systém sice vyžadoval vlastní verzi JVM, ale zdrojový kód zkompilovaný do bajtkódu byl vždy totožný. A jelikož je syntaxe podobná jazyku C, je pro programátory snažší na uchopení. [1]

1.2.1 Vybrané výhody Javy

Díky pochopitelné syntaxi je Java relativně lehká na naučení. Zároveň je také dostatečně robustní na to, aby v ní mohly být naprogramovány komplexní aplikace. Nevyžaduje od programátora správu paměti a tudíž tak eliminuje chybovost a zvyšuje spolehlivost programu. O paměť se stará automatický vyhledávač Garbage collector, který uvolňuje její nevyužívané části. Je vhodná pro tvorbu apletů díky své bezpečnosti a multiplatformovosti. Podporuje také vícevláknové programování, což umožňuje programu zpracovávat více úkolů zároveň. [1]

1.2.2 Objektově orientovaný jazyk

Hlavní jednotkou objektově orientovaného jazyku je právě objekt, což je v podstatě modul obsahující data a podprogramy. Objekt se nachází v určitém stavu, který reprezentují jeho data. K těm pak mají přístup podprogramy (operace), které jsou přístupné pro volání. Výhodou je, že takový objekt funguje jako takzvaná černá skříňka, která umí komunikovat s okolím, aniž bychom potřebovali znát její vnitřní funkci.

Důležitým pojmem je též dědičnost, kdy objekty jednoho druhu mohou přebírat schopnosti druhu jiného. V Javě se používají instance různých tříd (druhů objektů). Schopnost objektů různě reagovat na stejný typ příkazu se nazývá polymorfismus. Objekty obsahující stejná data co stejně reagují na stejné příkazy patří do stejné třídy. [2]

1.3 Důvod výběru Javy

Pokud beru v úvahu dvě nejznámější a tudíž i nejvíce zdokumentované platformy, zůstane ve výběru Java s Adobe Flash. I když je Flash spíše multimediální plat-

formou, je též dostatečně flexibilní a s použitím objektově orientovaného jazyku ActionScript se dá využít pro tvorbu apletů.

Z mého pohledu má ale ActionScript pořád blíže ke skriptovacímu jazyku, přičemž Java je plnohodnotným programovacím jazykem s delší historií. Má bohatou dokumentaci a u webové aplikace je velmi důležitým parametrem spustitelnost na různých rozdílných systémech, což splňuje dokonale. Nesmíme zapomenout ani na bezpečnost plynoucí z využívání bajtkódů. Ostatní výhody jsme si zmínili v podkapitole 1.2.1.

Poslední důvod je spíše praktický. Mám už s ní určité zkušenosti, které mohu prací na apletech dále rozvíjet. Nemusím se tedy učit naprosté základy a tím urychlím samotnou tvorbu.

Jediná nevýhoda Javy je pomalejší zavádění i těch nejjednodušších programů. V dnešní době výkonných elektronických zařízeních ale klady jasně převažují nad zápory.

2 APLET FÁZORŮ

2.1 Teorie

2.1.1 Dělení signálů

Signálem rozumíme fyzikální vyjádření zprávy, která může i nemusí být v elektrické podobě. Pokud je fyzikální veličinou napětí či proud, mluvíme o signálech elektrických. Ty a všechny jiné druhy lze rozdělit na dva základní typy, a to **deterministické** (určené) a **stochastické** (náhodné).

Budoucí průběh náhodného typu signálu nedokážeme s jistotou v přítomnosti určit. Lze ho pouze předpovědět s určitou pravděpodobností. Na druhou stranu určený signál s jistotou popsat umíme. Pro příklad může mít sinusový průběh či exponenciální charakteristiku. Tím se dostáváme k dalšímu rozdělení deteministického typu na **periodické** a **neperiodické**. [3]

Podle názvů dokážeme odvodit, že periodické jsou tvořeny opakováním určitého segmentu, což je třeba náš dříve zmíněný sinusový průběh. Platí pro ně vzorec:

$$s(t) = s(t + T_1). \tag{2.1}$$

Nejmenší hodnota T_1, pro kterou platí vzorec, se nazývá opakovací, nebo též základní perioda signálu. [4]

Neperiodický se v čase neopakuje. Takovým typem může být časový průběh elektrického napětí při nabíjení kondenzátoru. Periodický dále rozdělujeme na **harmonický** a **neharmonický**.

Z hlediska zaměření apletu nás zajímá pouze harmonický signál, který matematicky popisujeme funkcemi sinus a kosinus [3]:

$$u(t) = U_\mathrm{m} \cdot \cos(\frac{2\pi}{T_1}t + \varphi_1). \tag{2.2}$$

U_m ve vzorci je maximální hodnota výchylky – amplituda. Fázový posun mezi počátkem zvolené funkce (v našem případě $\cos x$) a počátkem souřadnic se nazývá počáteční fáze φ_1. Úhlový kmitočet pak odvodíme jako $\omega_1 = 2\pi f_1 = 2\pi/T_1$.

2.1.2 Komplexní vyjádření harmonického signálu

Pro komplexní vyjádření harmonického signálu využijeme Eulerovy vztahy:

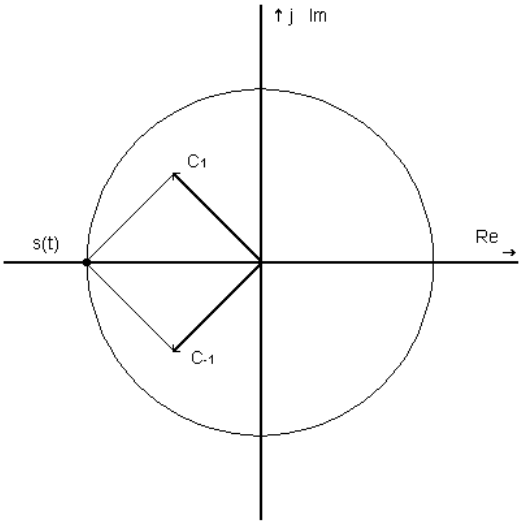
$$\cos x = \frac{\mathrm{e}^{\mathrm{j}x} + \mathrm{e}^{-\mathrm{j}x}}{2}, \ \sin x = \frac{\mathrm{e}^{\mathrm{j}x} - \mathrm{e}^{-\mathrm{j}x}}{2\mathrm{j}}. \tag{2.3}$$

Náš ukázkový signál pak pomocí vztahů můžeme přepsat:

$$\begin{aligned} s(t) &= C_1 \cos(\omega_1 t + \varphi_1) = C_1 \frac{\mathrm{e}^{\mathrm{j}(\omega_1 t + \varphi_1)} + \mathrm{e}^{-\mathrm{j}(\omega_1 t + \varphi_1)}}{2} \\ &= \frac{C_1}{2}\mathrm{e}^{\mathrm{j}\varphi_1}\mathrm{e}^{\mathrm{j}\omega_1 t} + \frac{C_1}{2}\mathrm{e}^{-\mathrm{j}\varphi_1}\mathrm{e}^{-\mathrm{j}\omega_1 t} = c_1\mathrm{e}^{\mathrm{j}\omega_1 t} + c_{-1}\mathrm{e}^{-\mathrm{j}\omega_1 t}. \end{aligned} \tag{2.4}$$

Dvě výsledné složky $c_1\mathrm{e}^{\mathrm{j}\omega_1 t}$ a $c_{-1}\mathrm{e}^{-\mathrm{j}\omega_1 t}$ se dají znázornit jako dva proti sobě rotující fázory. První fázor má poloviční velikost amplitudy signálu a rotuje proti směru hodinových ručiček rychlostí ω_1 radiánů za sekundu. Druhý rotuje se stejnou úhlovou rychlostí proti směru hodinových ručiček, přičemž má opačnou počáteční fázi. Jejich sčítání probíhá na reálné ose a výsledná velikost odpovídá velikosti harmonického signálu. Komplexní model zobrazuje obrázek 2.1.

Periodický signál obsahující více harmonických složek lze znázornit obdobně. Graf by se ale s větším počtem fázorů začínal znepřehledňovat, což u apletu vysvětlujícího podstatu funkce fázorů vhodné není. Proto nezobrazuje složku rotující proti směru hodinových ručiček a zobrazená rotující složka má výslednou velikost součtu.



Obr. 2.1: Komplexní model harmonického signálu

2.1.3 Fourierova řada periodického signálu

Každý periodický signál se skládá ze **stejnosměrné** a **střídavé složky.** Stejnosměrná odpovídá střední hodnotě signálu za periodu T. Střídavá složka má nulovou střední hodnotu a obsahuje první a vyšší harmonické, kterých je obecně nekonečný počet [4]. Frekvence první harmonické odpovídá opakovací periodě signálu T a frekvence vyšších harmonických jsou celočíselným násobkem první. Jednotlivé harmonické složky (Fourierovu řadu) můžeme získat operací zvanou Harmonická analýza, kterou navrhnul Jean Baptiste Fourier roku 1823. [3].

Fourierovu řadu můžeme zapsat v několika různých tvarech [3]. Buď rozkladem na sinusovou a kosinovou složku:

$$s(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos(k\omega_1 t) + b_k \sin(k\omega_1 t), \tag{2.5}$$

v kosinovém tvaru:

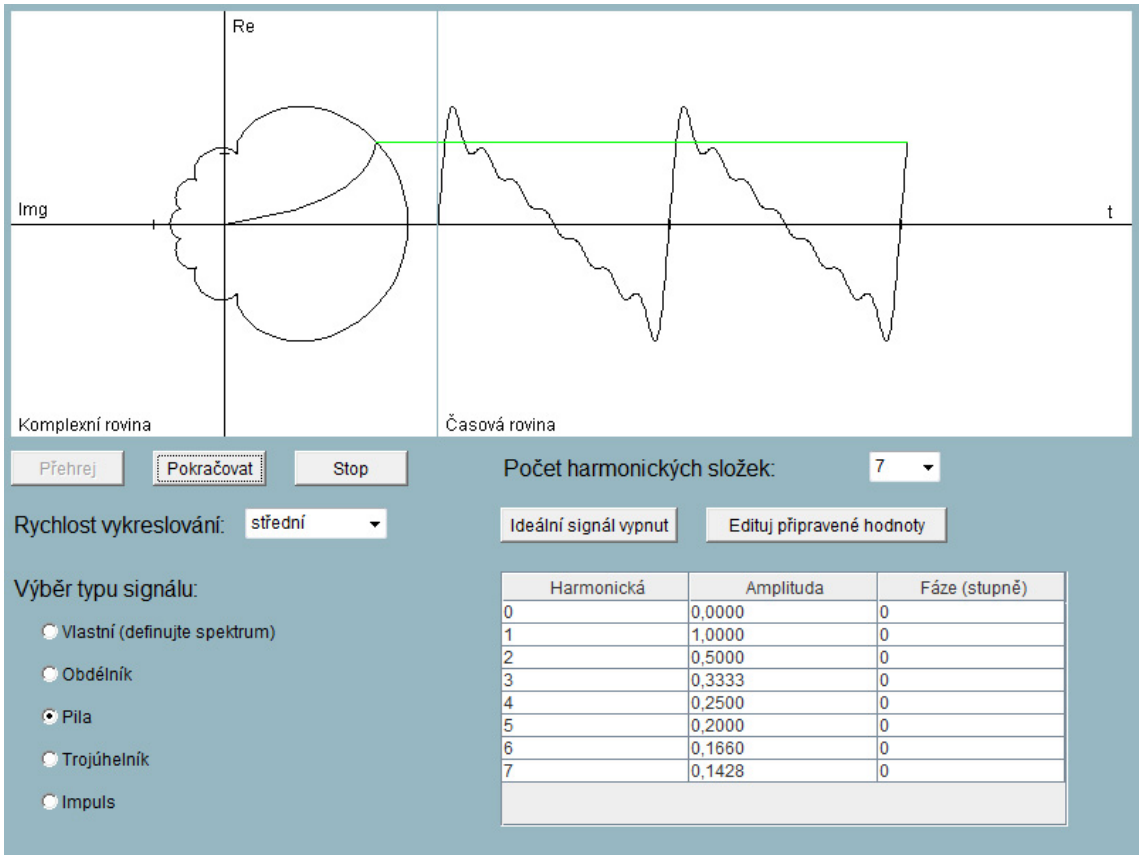
$$s(t) = \frac{C_0}{2} + \sum_{k=1}^{\infty} C_k \cos(k\omega_1 t + \varphi_k), \tag{2.6}$$

případně v komplexním tvaru pak:

$$s(t) = \frac{C_0}{2} + \sum_{k=-\infty}^{\infty} c_k \mathrm{e}^{\mathrm{j}k\omega_1 t}. \tag{2.7}$$

2.2 Popis apletu

Na obrázku 2.2 se nachází ovládací rozhraní apletu tak, jak jej vidíme při načtení do internetového prohlížeče. Všechny aplety byly otestovány v prohlížeči Google Chrome ve verzi 35.0.1916.114 m, Opera ve verzi 12.17 a Firefox ve verzi 29.0.1.



Obr. 2.2: Okno apletu fázorů

Základní část apletu tvoří dvojice grafů, nacházející se v horní části okna. Levý graf znázorňuje komplexní rovinu, kde probíhá vykreslování vybraných fázorů. Pravý širší graf znázorňuje časovou rovinu, kam probíhá vykreslování výsledného signálu v čase.

2.2.1 Komplexní rovina

Podle obrázku 2.1 víme, že vodorovná rovina na níž probíhá součet fázorů je reálná, zatímco svislá imaginární. To by ovšem znamenalo, že výsledná amplituda signálu by se promítala na vodorovnou reálnou osu a tudíž by se samotný signál v čase vykresloval svislým směrem.

V širokém spektru publikací je ovšem zvyklostí průběh signálu zakreslovat na vodorovnou časovou osu, přičemž směr je dán z levé strany do pravé. Cílem apletu

je ale posloužit studentům jako přehledný a interaktivní nástroj pro vysvětlení funkce fázorů. Proto bylo rozhodnuto celý graf komplexní roviny pootočit o 90 stupňů proti směru hodinových ručiček. Tím je možné zachovat veškeré zvyklosti při vykreslování časové osy.

Otočením os by se ovšem změnil i počáteční úhel vykreslování. Znamenalo by to, že by se posunula i počáteční fáze vykreslovaného signálu a tím by se jeho průběh posunul směrem doleva. Předpřipravený obdélníkový signál by tak nezačínal nástupnou hranou. Proto zůstala pozice nulového úhlu nezměněna.

Jak už bylo zmíněno v teorii, v komplexní rovině se zobrazují dva proti sobě rotující fázory s poloviční velikostí amplitudy, ne ovšem v apletu. Jedná se o kompromis mezi přesností a přehledností. Na grafu tak vidíme jednu složku rotující proti směru hodinových ručiček s výslednou velikostí amplitudy.

Výsledný součet fázorů za sebou zanechává stopu, která se při každém zastavení vymaže.

Značky na reálné i imaginární ose naznačují rozměry jednotkové kružnice, která vznikne vykreslením jednoho fázoru s amplitudou 1.

2.2.2 Časová rovina

Na časové ose se vykresluje výsledek součtu všech zadaných fázorů v čase. Výška amplitudy přesně odpovídá aktuálnímu součtu na reálné ose v komplexní rovině. Přesně to znázorňuje vodorovná zelená linka, která prochází oběma rovinami, takže uživatel mezi nimi jasně vidí vztah.

Značky na ose naznačují tři periody signálu, které se do grafu vykreslí.

2.2.3 Ovládání vykreslování a rychlosti

Tlačítko Přehrej slouží ke spuštění vykreslování. Dokud se automaticky nevykreslí tři periody nebo dokud uživatel vykreslování nezastaví tlačítkem Stop, je toto tlačítko neaktivní.

Uživatel je schopen vykreslování kdykoliv pozastavit a poté znovu spustit. Mezitím může například změnit počet vykreslovaných harmonických složek a na časové ose pak uvidí, jak se signál blíží ideálnímu průběhu. K tomuto účelu slouží tlačítko Pauza, které se po stisknutí změní na Pokračovat. Tlačítko Stop pak slouží k předčasnému zastavení vykreslování.

Pokud se uživateli zdá vykreslování příliš pomalé nebo rychlé, může si jeho rychlost zvolit roletkou Rychlost vykreslování. Toto nastavení má lehký vliv na stopu v komplexní rovině. Při pomalé rychlosti je stopa viditelně vyhlazenější. Na tvar samotné stopy ale vliv nemá.

2.2.4 Typy signálů

Aby uživatel nemusel ztrácet čas vkládáním vlastních dat, může si v apletu vybrat už předpřipravené hodnoty. K dispozici jsou základní průběhy jako obdélníkový, pilový a trojúhelníkový. Posledním typem je periodicky se opakující impuls.

2.2.5 Ovládání tabulky

Parametry vybraného typu průběhu se zobrazují v tabulce, umístěné na pravé straně apletu. První sloupec obsahuje pořadové číslo harmonické. Frekvence každé složky je tedy celočíselným násobkem základní s pořadovým číslem 1. Číslo 0 značí stejnosměrnou složku. Pokud ji hodláme použít, pro správné zobrazení je nutné její počáteční fázi nastavit na 90 stupňů (kvůli otočeným osám v komplexní rovině). Druhý sloupec zobrazuje amplitudu složky. Díky značkám na osách komplexní roviny můžeme odhadnout její reálnou velikost na grafu. Přípustná jsou čísla menší i větší než 1. Je nutné parametr vkládat s desetinou tečkou. Nulová hodnota automaticky složku vyřazuje ze součtu. Třetí sloupec zobrazuje počáteční fázi složky. Zadává se ve stupních a 0 označuje pravou stranu imaginární osy. Stupně se přičítají proti směru hodinových ručiček.

Pokud uživatel vybere předpřipravený signál, tabulka slouží pouze ke zobrazování hodnot a nelze v ní provádět změny. Při výběru vlastního typu signálu se tabulka vynuluje a odemkne pro zadávání. K dispozici je 16 harmonických složek, přičemž první značí stejnosměrnou složku. Pokud budou do kolonky vložena data nevhodného typu, vykreslování spustit nepůjde, dokud chyba nebude opravena.

Aby bylo možné upravovat připravená data a pozorovat tak vliv změn na součet fázorů, je nad tabulkou umístěno tlačítko Edituj připravené hodnoty. Stane se aktivním v případě, že je právě přehráván připravený signál. Po stisku překopíruje tato data do prázdné tabulky, kde se hodnoty odemknou pro editaci.

Každý připravený typ signálů má v programu definován svůj ideální průběh. Ten lze libovolně zapínat a vypínat tlačítkem Ideální signál vypnut/zapnut. Lze tak pozorovat nepřesnost konečného počtu složek.

Nad tabulkou se dále nachází roletka s výběrem počtu zobrazovaných harmonických složek. Spolu s tlačítkem Pauza si může uživatel zobrazit rozdíl mezi vysokým a nízkým počtem složek.

2.3 Interní funkce apletu

Aplet je celý napsán ve veřejné třídě Fazory, která je rozšířena o třídu Applet a implementuje rozhraní Runnable, ActionListener a ItemListener.

2.3.1 Prvotní inicializace

Veškerou inicializaci provádí metoda init(). Probíhá zde samotné grafické nastavení okna, tlačítka jsou přidávána a umisťována na vhodné pozice. Při výběru vhodného layoutu jsem nebyl s žádným rozložením ovládacích prvků plně spokojen. Jelikož aplet není nijak komplexní a velikost okna zůstane neměnná, rozhodl jsem se layout vůbec nevyužívat a rozmisťovat prvky do absolutních pozic nastavením layoutu na prázdnou hodnotu.

```
setLayout(null);
```

Dále se v metodě definuje tabulka, která je instancí třídy JTable, vkládá se do kontejneru JScrollPane přidávající posuvníky, nastavují se proměnné na prvotní hodnoty a probíhá zde i vykreslování os veškerých grafů a příprava obrázkových bufferů.

2.3.2 Vykreslování fázorů

Uvnitř metody run() se nachází základní smyčka programu. Jelikož programovaný aplet vykresluje v čase graf, bylo nutné zajistit plynulost a dostatečně pomalý běh obnovování okna. Z tohoto důvodu je v programu vytvořena instance třídy Thread, která definuje jedno vlákno procesu. Thread se používá hlavně u komplexnějších aplikací, kde paralelně běží více výpočtů a tedy i více vláken. Zde je využíván hlavně pro tvorbu zpoždění. V metodě init() se vypočítá potřebné zpoždění pro zadaný počet snímků za vteřinu (v tomto případě 60). Na konci metody run() se pak vlákno pozastaví metodou sleep, čímž vytvoříme smyčku spouštějící se šedesátkrát za vteřinu.

Smyčka dále obsahuje kód, řídící běh vykreslování. Využívá se k tomu hlavně dvojice proměnných datového typu boolean s názvy isRendered a isPaused. Na tomto místě také probíhá sčítání fázorů. Kód je velice jednoduchý a probíhá ve smyčce while.

```
endX   =  (startX + (modules[chosenSingal][counter]*50)
* Math.sin(counter2*angle+1.570796+phase[chosenSingal][counter]));

endY   =  (startY + (modules[chosenSingal][counter]*50)
* Math.cos(counter2*angle+1.570796+phase[chosenSingal][counter]));

backgBuffer.drawLine((int)startX, (int)startY, (int)endX, (int)endY);

startX = endX;
startY = endY;
```

Každý fázor definuje počáteční a koncový bod. Při první iteraci smyčky while se jako počáteční bod považuje střed grafu na souřadnicích x = 150 a y = 150. Z připravené řady hodnot s názvem modules se k počátku přičte délka fázoru, která se následně vynásobí vhodnou funkcí, přičemž se použije hodnota úhlu (který se iteruje mimo tuto smyčku) vynásobená pořadovým číslem harmonické složky. Tím zajistíme, že frekvence harmonických složek odpovídá celočíselným násobkům základního kmitočtu. K úhlu dále přičítáme počáteční fázi z definovaného pole phase. Výsledkem jsou koncové souřadnice zpracovávaného fázoru. Následně se fázor vykreslí do bufferu, který je instancí třídy Image.

Takto smyčka while proběhne přesně tolikrát, kolik složek máme vybráno v roletce Počet harmonických složek. Koncový bod předchozího fázoru se použije jako počáteční bod následujícího, čímž se do obrázkového bufferu vykreslí všechny vybrané fázory. Na konci iterace jsou v proměnných endX a endY uloženy souřadnice koncového bodu posledního fázoru v řadě, což je grafický výsledek součtu. Následující řádek kódu vykresluje časový průběh:

```
backgBuffer.drawLine((int)(25.942*(angle-speed)),
(int)endYprev,(int)(angle*25.942), (int)endY);
```

V dalším kroku se využije pouze proměnná endY, která nese výslednou výšku amplitudy časového průběhu. V grafu časové roviny se osa posouvá díky společné proměnné angle. Hodnota z radiánů je přepočítána na pixely, čímž dostaneme druhou souřadnici bodu. Tento bod se pak propojí linkou s předchozím. Jelikož aplet vykresluje buffer šedesátkrát za vteřinu, je časový průběh signálu plynulý.

Pro usnadnění výpočtů se namísto stupňů používají radiány. Jelikož ovládací rozhraní tabulky pracuje se stupni, je v programu vytvořen přepočet mezi těmito jednotky.

2.3.3 Zamezení problikávání

Pokud by se veškeré změny v grafech nevykreslovaly do bufferu, docházelo by k nepříjemnému blikání pohybujících se čar. Tomu je v apletu zamezeno třemi způsoby. První je už dříve popsané využití třídy Thread, která vlákno na vhodné doby uspí tak, aby se vytvořila konstantní vykreslovací frekvenci. Dále je nutné veškeré operace vykreslovat do speciálně vyhrazených obrázků, které jsou v kódu nazývány buffery. Místo metody paint() je také využívána metoda repaint(), která volá metodu update(), ve které se nachází zmíněné buffery.

3 APLET FILTROVÁNÍ AUDIO SIGNÁLŮ

3.1 Teorie

3.1.1 Fourierova transformace

Fourierova transformace (FT) patří k základním metodám analýzy signálů. Slouží pro převod signálu z časové oblasti do frekvenční. Výsledkem jsou pak spektra, ve kterých identifikujeme velikosti a frekvence (frekvenční modulové spektrum) nebo fáze jednotlivých složek signálu (frekvenční fázové spektrum). Pohled do spekter nám umožňuje získat lepší přehled o složení signálu, zjistit rozložení energie nebo se dozvědět, jakou šířku spektra zabírá. Tyto znalosti pak můžeme aplikovat při dalším zpracování signálu. FT se proto využívá při zpracování audio signálů, pro analýzu datových kanálů, pro rozpoznání vzorů v obrazech, v lékařské diagnostice, v optice a nesčetně mnoha dalších aplikacích.

Podmínkou FT je, aby byla transformovaná funkce $s(t)$ absolutně integrovatelná v intervalu $-\infty, \infty$. Spojitou Fourierovu transformaci udává vztah:

$$S(\omega) = \int_{-\infty}^{\infty} s(t) \mathrm{e}^{-\mathrm{j}\omega t} \mathrm{d}t, \tag{3.1}$$

inverzní je pak definována:

$$s(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} S(\omega) \mathrm{e}^{\mathrm{j}\omega t} \mathrm{d}\omega. \tag{3.2}$$

ω značí úhlový kmitočet $\omega = 2\pi f$.

3.1.2 Diskrétní FT

Diskrétní FT je jedna z nejrozšířenějších metod FT. Využívá se například pro zpracování digitálních navzorkovaných signálů periodických i neperiodických. Vstupem je vždy konečná posloupnost vzorků s_k a výstupem pak konečná posloupnost komplexních koeficientů DFT X_n. Transformuje se tedy vždy jen omezená délka vzorkovaného signálu. Pro výpočet N spektrálních čar z N vzorků signálu se použije tento předpis:

$$X_n = \sum_{k=0}^{N-1} s_k \mathrm{e}^{-\mathrm{j}kn2\pi/N}. \tag{3.3}$$

Výstupem DFT je pouze informace o zastoupení jednotlivých harmonických složek a jejich velikostí, nikoliv jejich lokalizace v čase. Proto mohou mít stejná spektra dva signály, které se v časovém prostoru odlišují.

3.1.3 FFT

DFT pro praktické užití ovšem příliš vhodná není, neboť její výpočetní náročnost roste se čtvercem délky vektorů $O(N^2)$. V praxi se tedy častěji používá algoritmus rychlé Fourierovy transformace (FFT), který byl publikován v roce 1965. FFT výrazně redukuje výpočetní dobu minimalizací počtu násobení. Vyžaduje ovšem, aby počet bodů N odpovídal celočíselné mocnině čísla 2. Časová náročnost výpočtu je pak $O(N \ln N)$. [8]

3.1.4 Kmitočtové filtry

Kmitočtový filtr je představitelem lineárního neparametrického systému, který určitou část spektra signálu přenáší, a jinou naopak potlačuje. Využívá se v mnoha odvětvích elektrotechniky, jako například v radiotechnice, telekomunikacích, elektroakustice či hudební technice.

Prakticky mohou být filtry realizovány z diskrétních součástek (kondenzátor, odpor, cívka) nebo z integrovaných elektrických obvodů. Speciálním druhem jsou pak číslicové filtry, jejichž výhodou oproti analogovým je větší přesnost a univerzálnost. Obecně můžeme filtry rozdělit na selektivní, korekční a fázovací obvody. [4]

3.1.5 Selektivní filtry

Tyto filtry mají za úkol potlačovat spektrum signálu v nepropustném pásmu a propouštět signál v propustném. Podle tvaru modulové kmitočtové charakteristiky filtry dělíme na čtyři základní typy.

Dolní propust (DP) propouští celou část kmitočtového spektra, které se nachází pod definovanou mezní frekvencí. **Horní propust** (HP) propouští část spektra nacházející se nad mezní frekvencí. **Pásmová propust** (PP) propouští pásmo definované středním kmitočtem a šířkou pásma. **Pásmová zádrž** (PZ) naopak potlačuje pásmo definované středním kmitočtem a šířkou pásma. V praxi se používají i jiné druhy filtrů. Jedním z nejpoužívanějších je například hřebenový filtr.

3.1.6 Korekční filtry

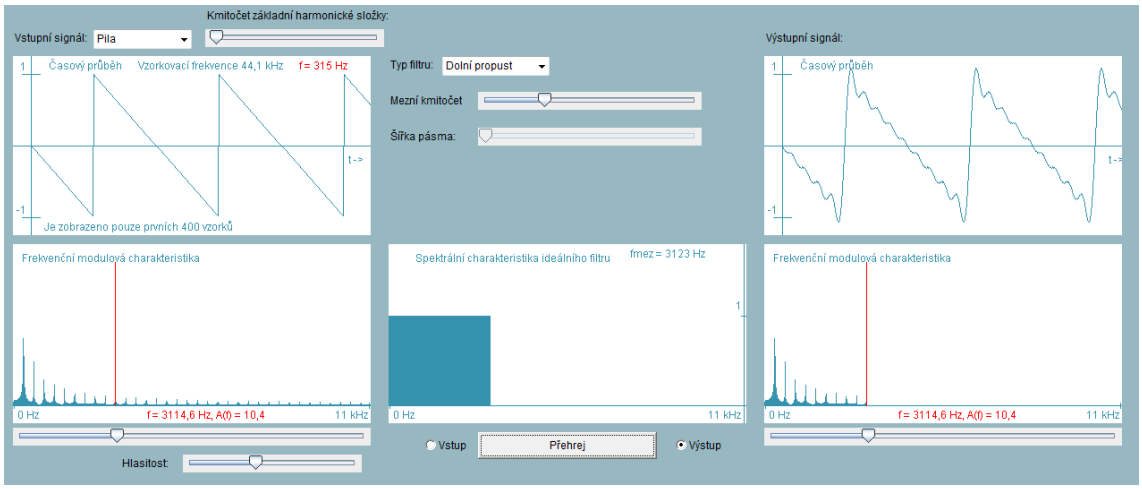
Korekční filtry mají za úkol korigovat přenos určitých bloků přenosového řetězce, aby byl přenos celé soustavy kmitočtově nezávislý.

3.1.7 Fázovací obvody

Tyto filtry neslouží k filtraci signálu skrze kmitočtově závislou frekvenční modulovou charakteristiku, nýbrž využívají kmitočtově závislou frekvenční fázovou charak-

teristiku. Využívají se tam, kde je třeba korigovat fázový posun složek na základě kmitočtu.

3.2 Popis apletu



Obr. 3.1: Okno apletu kmitočtové filtrace

3.2.1 Sloupec vstupního signálu

Okno apletu je přehledně rozděleno do tří sloupců. První je vstupní řetězec zvukového systému. Z roletky je možné si vybrat harmonické signály a sliderem Kmitočet základní harmonické složky zvolit tento kmitočet. Přítomny jsou též dva šumy (bílý a růžový) a čtveřice zvukových ukázek (tekoucí voda, štěkot psa, výstřel a brum starého televizoru). Všechny vstupní signály jsou navzorkovány frekvencí 44,1 kHz.

Graf časového průběhu zobrazuje vzorky vybrané zvukové ukázky. Pokud uživatel vybere harmonické signály či šumy, zobrazuje se pouze 400 prvních vzorků. Kdyby byla zobrazena obálka celé ukázky, nebylo by například vidět, jak potlačení vysokých harmonických složek deformuje tvar obdélníkového signálu. U ostatních ukázek už to tak nevadí, takže se zobrazují jako amplitudová obálka celé jedné vteřiny.

Graf frekvenční modulové charakteristiky pak zobrazuje spektrum signálu až do 11 kHz. Slider pod grafem pohybuje ukazatelem, který vypisuje frekvenci a velikost vybraného modulu. Je tedy možné si třeba orientačně změřit dominantní složky zvuků či frekvence složek harmonických signálů. Stejný slider je i u grafu výstupní modulové charakteristiky a oba jsou provázány.

V prvním sloupci se také nachází slider ovládání hlasitosti.

3.2.2 Sloupec filtru

V druhém sloupci se nachází roletka s výběrem filtru. Obsahuje ideální verze dolní, horní a pásmové propusti a pásmové zádrže. Dvěma slidery je možné v případě DP a HP měnit mezní frekvenci až do 11 kHz. V případě PP a PZ se jedná o střední frekvenci a šířku pásma.

Je důležité zmínit, že všechny zvukové ukázky jsou vzorkovány frekvencí 44,1 kHz, ale veškeré grafy jsou kvůli názornosti omezeny 11 kHz. Zbytek spektra do 22,05 kHz není zobrazen, ale při filtrování se pořád uplatňuje. Pokud tedy například uživatel vybere DP a nastaví mezní frekvenci na maximum (11 kHz), je signál potlačován od 11 Khz až do 22,05 kHz.

Na konci sloupce je umístěno tlačítko pro spuštění a zastavení přehrávání vybraného signálu. Je samozřejmě možné vybrat vstupní či výstupní signál a akusticky porovnat výsledek filtrace. Přehrávání není nutné při změně jakéhokoliv parametru v apletu zastavovat. Veškeré provedené změny se ihned projevují.

3.2.3 Sloupec výstupního signálu

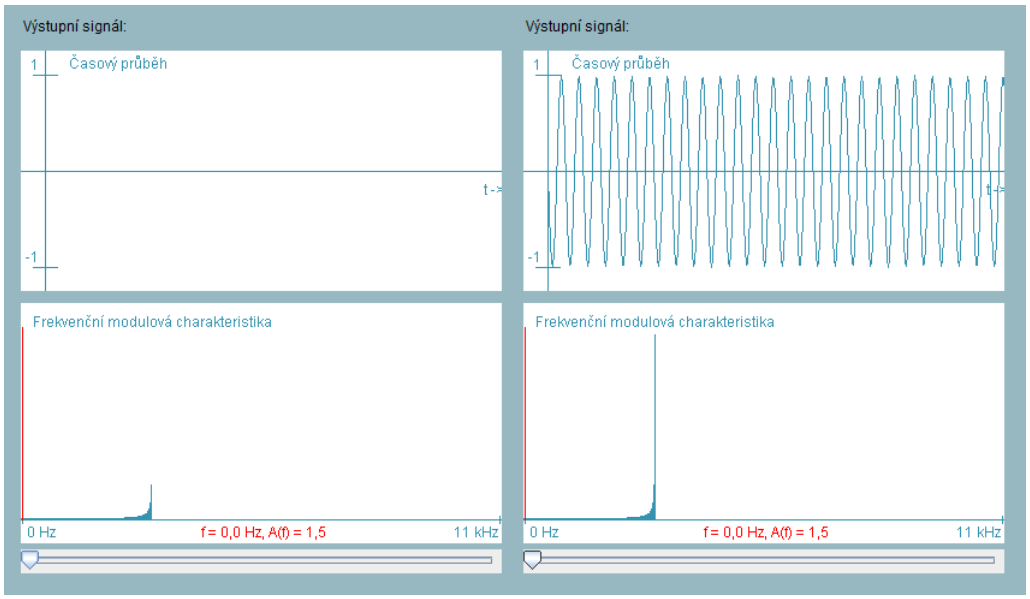
Tento sloupec vypadá obdobně, jako první. V prvním grafu je zobrazován časový průběh výstupního signálu a platí pro něj stejné vlastnosti, jako pro graf časového průběhu vstupního signálu (prvních 400 vzorků pro harmonické signály a šumy, celá obálka pro ostatní zvukové ukázky). Při bližším pohledu se může zdát, že vykreslování časového průběhu se chová nestandardně. Změny při některých filtrací se projevují skokově, jak například filtrování sinusového průběhu, což ukazuje obrázek 3.2. Graf je prázdný do té doby, než se modul s největší velikostí přesně dostane do propustného pásma. Faktem ovšem je, že signál narůstá postupně. Jen je tento nárůst tak malý, že v grafu prostě není vidět.

Druhý graf zobrazuje frekvenční modulovou charakteristiku výstupního filtrovaného signálu. Slider ukazatele pod grafem je provázán s prvním ukazatelem.

3.3 Interní funkce apletu

3.3.1 Vstupní signál

Vzorky vstupního signálu jsou do apletu zavedeny několika různými způsoby. Periodické signály (sinus, pila, obdélník) aplet sám generuje. Vstupním parametrem je vybraná frekvence základní harmonické složky. Funkce generateSin(int inputFreq) generuje 1 vteřinu sinusového signálu vzorkovacím kmitočtem 44,1 kHz. Pila a obdélník se generují obdobným způsobem, jako je uvedený kód pro sinus.



Obr. 3.2: Skokové změny časového průběhu při filtraci sinu

```
float value = 44100.0f/inputFreq;
for( int i=0; i<dataInput.length; i++ ){
   dataInput[i] = (float) Math.sin(Math.toRadians((360.0/value)*i));
   }
```

Vzorky šumů byly vygenerovány programem Audacity a následně uloženy jako 32 bitové mono stopy ve formátu wav. Tyto soubory jsou uloženy v balíčku jar, ze kterého je aplet načítá. Pro načtení se používá funkce loadFloatSample externí knihovny JSyn. Následující řádek kódu načítá vzorek bílého šumu.

```
mySample = SampleLoader.loadFloatSample(
           getClass().getResource("/white.wav"));
```

Stejným způsobem jsou načítány i ostatní zvukové ukázky, které byly získány ze serveru freesound.org. Na soubory se vztahuje licence Creative Commons 0. Následně jsou vzorky z pole dataInput rozkopírovány do dalších dvou polí realDataInput a realDataInputBack, a to převážně z toho důvodu, že funkce diskrétní Fourierovy transformace je tzv. „in place“, což znamená, že výstup transformace přepíše vstupní data, které nicméně potřebujeme pro přehrání vstupu.

3.3.2 Realizace DFT

Z důvodu značné úspory času psaní kódu je v apletu využita externí knihovna JTransforms, která realizuje třídy pro různé druhy transformací. V apletu je využita pouze FFT a inverzní FFT. Jakmile jsou data připravena, provede se funkce

rffFFT() a uvnitř dvě transformace. Výstupem první je 800 spektrálních čar, přičemž polovina se zobrazuje v grafu frekvenční modulové charakteristiky. Výstupem druhé je vyšší počet modulů pro přesnější provedení filtrace. Při inverzní transformaci tak získáme zpět požadovaný počet vzorků pro přehrání.

```
dfft.realForward(dataInput);
dfft2.realForward(realDataInput);
```

Po provedení filtrace se funkcí rbfFFT() provádí pouze jedna inverzní transformace.

```
dfft2.realInverse(realDataOutput, true);
```

3.3.3 Aplikace filtru

Filtrace se provádí skrze pole realFilterChar, jehož velikost odpovídá polovině velikosti pole realDataOutput, kde se nacházejí výstupní koeficienty FFT. To je dvojnásobně veliké, neboť kromě reálných koeficientů obsahuje i imaginární. Dle nastavené mezní/střední frekvence a šířky pásma se do pole filtru zapíší jedničky na místa odpovídající frekvencím, které chceme propustit. Nuly se zapíší do míst, kde signál potlačíme. Uvedený kód patří dolní propusti. Ostatní typy fitrů jsou řešené obdobně.

```
for(int i=0; i<realFilterChar.length; i++){
   if(i<= filterFreqSlide.getValue()){realFilterChar[i]=1; }
   else{realFilterChar[i]=0; }
}
```

Následně se pole realFilterChar a realDataOutput vynásobí.

```
for (int i=1; i<realFilterChar.length; i++){
   realDataOutput[i*2]= realDataOutput[i*2]*realFilterChar[i];
   realDataOutput[i*2+1]=realDataOutput[i*2+1]*realFilterChar[i];
}
```

Nakonec se provede už zmíněná inverzní DFT.

3.3.4 Přehrávání vzorků

Pro přehrání vstupních a výstupních vzorků je v apletu použita externí knihovna JSyn. Jedná se o robustní modulární syntezátor, který umožňuje prokročile pracovat se zvukovým signálem. V apletu funguje pouze jako přehrávač. Základem je instance syntezátoru, do které se připojí výstup a přehrávač. Následně se přehrávač připojí na výstup a syntezátor se spustí.

```
synth = JSyn.createSynthesizer();
out= new LineOut();
synth.add(out);
synth.add(samplePlayer = new VariableRateMonoReader());
samplePlayer.output.connect( 0, out.input, 0);
samplePlayer.output.connect( 0, out.input, 1);
synth.start();
```

Spuštění či zastavení přehrávání se řeší spuštěním či zastavením výstupu.

```
out.start();
out.stop();
```

Zda bude přehráván vstupní zvukový signál či filtrovaný signál řeší fronta přehrávače samplePlayer, do níž se vkládá vstupní či výstupní vzorek.

```
samplePlayer.dataQueue.queueLoop(mySample);
samplePlayer.dataQueue.queueLoop(mySampleOutput);
```

4 APLET UMĚLÉHO BARVENÍ ŠEDOTÓNOVÝCH OBRAZŮ

4.1 Teorie

Černobílé snímky mohou být výstupem rentgenového měření nebo sonografického vyšetření. Nevztahují se pouze k lékařské diagnostice. Šedotónové snímky (z anglického slova grayscale) jsou také výstupem termografického měření, které zaznamenává infračervené vyzařování tělesa. Termovizní systémy mají mnoho uplatnění, například v armádě, kde se termografie používá pro detekci lidského tepla či v naváděcích systémech raket. V civilním sektoru se termovize využívá například při hledání úniků tepla z budov.

Pokud je výstupem měření šedotónový snímek, je pravděpodobné, že se využije i umělé barvení. Pozorovatel je totiž v průměru citlivější na odstíny barvy, než na odstíny šedi [5]. V barevném obrázku tudíž může při diagnostice zaznamenat více detailů a rychleji a spolehlivěji tak určit diagnózu (v případě lékařské diagnostiky), nebo rychleji vyhledat chtěné informace (např. únik tepla). Pro barvení se zpravidla využívá flexibnilní softwarové řešení, i když i hardwarové mělo své využití. Je možné použít několik metod.

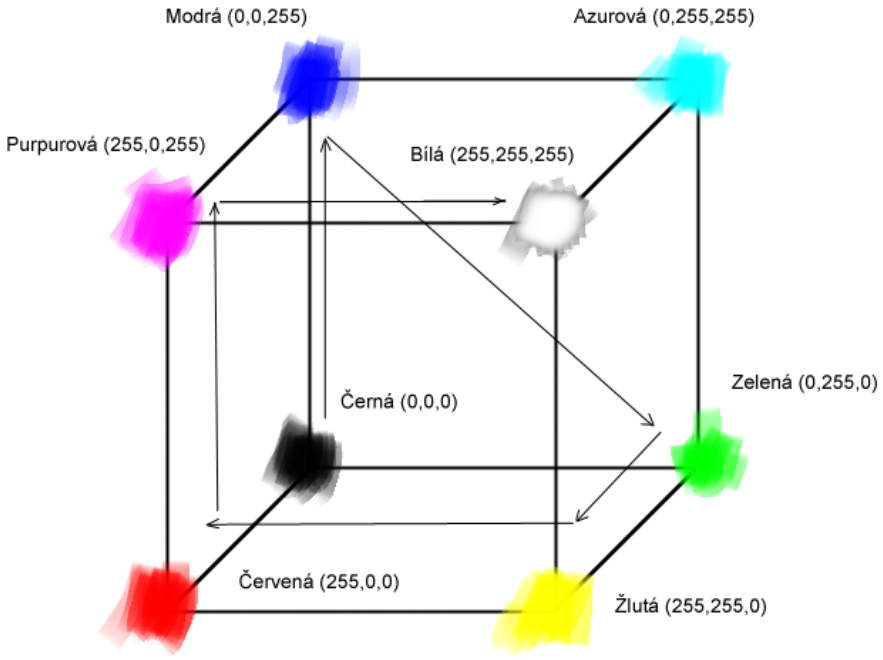
4.1.1 Metoda parametrické křivky

Barevný model RGB můžeme zobrazit jako krychli [7]. V každém rohu se nachází taková kombinace třech složek RGB, jejichž hodnota je buď nulová, nebo maximání. Jelikož model RGB pracuje se třemi složkami, je možných kombinací 2^3, což přesně odpovídá počtu rohů krychle. Pokud budeme barvu reprezentovat 24 bity, pak bude krychle vypadat jako na obrázku 4.1.

V tomto prostoru pak můžeme barvení popsat křivkou, vytvořenou z ekvidistantně rozložených bodů. [5] Souřadnice těchto bodů pak odpovídají jednotlivým stupňům šedi vstupního obrázku. Aby byly zachovány kontinuální změny jasu, musí mít křivka spirálovitý tvar, jejíž konce se nacházejí v rozích, představující bílou a černou barvu.

4.1.2 Metoda Schrödingerovy krychle

Tato metoda též pracuje s RGB krychlí. Vstupní rozsah jasu (0–255) se rozdělí do šesti částí, přičemž tři výstupní složky RGB v každém segmentu soupají, klesají či zůstávají na minimu nebo maximu. V krychli se toto chování dá vyznačit směrem z černé do modré, z modré do zelené, ze zelené do žluté, z žluté do červené, z červené do purpurové a z purpurové do bílé, jak je vyznačeno v obrázku 4.1. [5]



Obr. 4.1: RGB krychle s vyznačeným algoritmem Schrödingerovy krychle

4.1.3 Metoda barevné palety

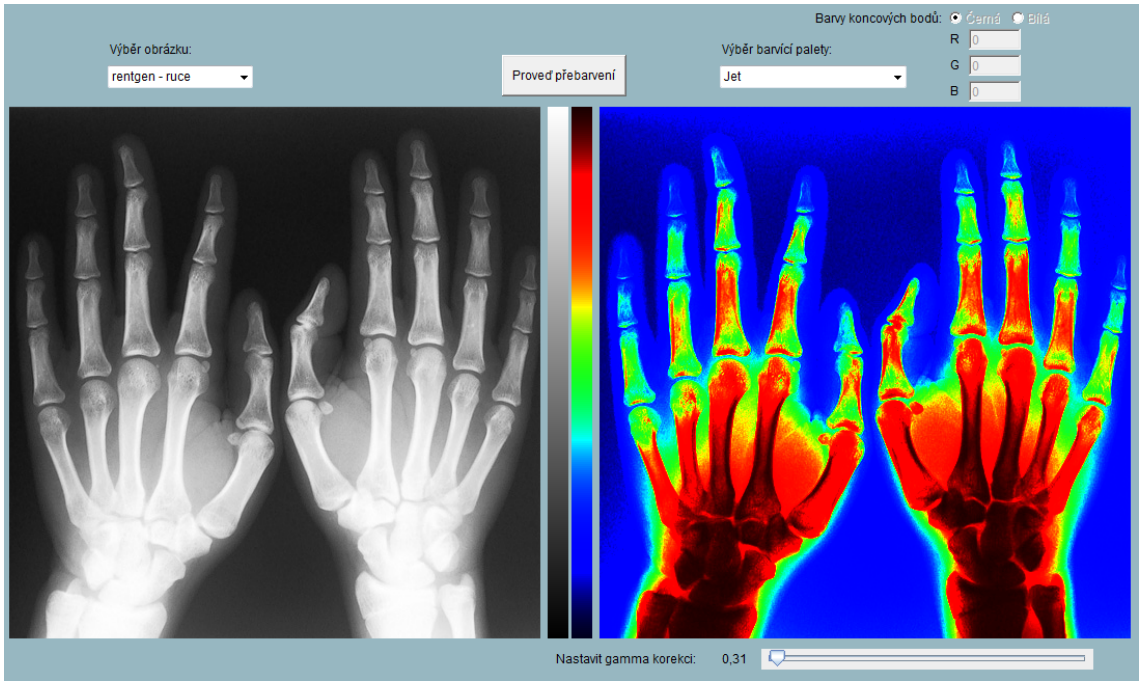
Jsou vytvořeny různé barvící palety, které mají formu matic. Tři řádky představují složky R, G a B. Každý sloupec je pak vstupní hodnotou jasu. Algoritmus analyzuje obrázek pixel po pixelu a dle vstupní hodnoty vyhledá ve vybrané paletě příslušné složky, z nichž vytvoří výstupní barvu. Tato metoda je poměrně jednoduchá na implementaci a je využita ve vytvořeném apletu.

4.1.4 Gama korekce

Lidské oko nevnímá jas lineárně, nýbrž logaritmicky. Jas starších CRT monitorů byl naopak zobrazován mocninou vstupního napětí elektronového děla, takže naše vnímání takového obrazu bylo lineární [7]. LCD monitory ovšem takovou závislost výstupního jasu nemají, a tudíž ji emulují přepočtem. Naše vnímání jasu ale ovlivňují i jiné faktory, jako například množství okolního světla. Proto se nám může obraz na obrazovce zdát moc tmavý, či přesvětlený. Abychom se linearitě přiblížili, je možné na monitorech měnit parametr γ.

Stejný přepočet je možné provést ještě před zobrazením obrazu na monitoru, což aplet umožňuje. Parametr gamma korekce lze nastavit od hodnoty 0,1 po 8,0.

4.2 Popis apletu



Obr. 4.2: Okno apletu umělého barvení

V levé části okna se nachází šedotónový obraz, který si uživatel může vybrat z roletky Výběr obrázku. K dispozici je několik rentgenových snímků, tři testovací obrazy a dvě fotky. V pravé části okna se nachází obraz, na který byla aplikována vybraná barevná paleta.

K výběru jsou předpřipraveny čtyři barevné palety HSV, Jet, cool a hot skrze roletku Výběr barvící palety. Jejich hodnoty jsou vyexportovány z Matlabu. Dále je možné vytvořit vlastní paletu výběrem položky Lineární gradient. Tím se odemknou políčka s názvem Barvy koncových bodů. Přepnutím na bílou a černou může uživatel zadat hodnoty RGB, které budou náležet koncům šedotónové palety. Ostatní hodnoty mezi těmito body se automaticky dopočítají. Následně je možno novou paletu aplikovat tlačítkem Proveď přebarvení. Ukázky šedotónové a barevné palety se nacházejí mezi vstupním a výstupním obrazem.

Pod výstupním obrazem se nachází slider Nastavit gamma korekci, který nastavuje výstupní gamma funkci barevné palety. Standardně je tato hodnota nastavená na 1,0, minimum je pak 0,1 a maximum 8,0. Při každé změně palety se hodnota resetuje zpět na 1,0.

4.3 Interní funkce apletu

Aplet pro barvení využívá metodu barevné palety. V programu je proto definováno několik fixních palet s názvy Jet, cool a hot. Hodnoty jejich složek RGB byly vyexportovány z programu Matlab. Paleta HSV je v apletu přímo generována změnou odstínu (parametr Hue) v barevném modelu HSV.

```
public void generateHSBpalette(){
  float hue = 0.0f;
  xCounter = 0;
while (xCounter <= 255){
  Color hsbC = Color.getHSBColor(hue, 1, 1);
  pallet[0][xCounter] = hsbC.getRed();
  pallet[1][xCounter] = hsbC.getGreen();
  pallet[2][xCounter] = hsbC.getBlue();
  xCounter++;
  hue = hue + 1.0f/255.0f;}
xCounter = 0;}
```

4.3.1 Aplikace gamma korekce

Na vybranou paletu se před barvením aplikuje gamma korekce. Všechny hodnoty všech RGB složek se přepočítají podle následujícího vzorce:

$$nováHodnota = 255 \times ((staráHodnota/255)^{1/gamma}). \tag{4.1}$$

Parametr gamma je v tomto případě číslo od 0,1 do 8,0. Výše uvedenému vzorci odpovídá následující kód.

```
palettGamma[0][xCounter]=(int)(255*Math.pow(
(pallet[0][xCounter]/255.0), 1/sliderToFloat[gammaCh.getValue()]));

palettGamma[1][xCounter]=(int)(255*Math.pow(
(pallet[1][xCounter]/255.0), 1/sliderToFloat[gammaCh.getValue()]));

palettGamma[2][xCounter]=(int)(255*Math.pow(
(pallet[2][xCounter]/255.0), 1/sliderToFloat[gammaCh.getValue()]));
```

4.3.2 Barvení obrazu

Aplet paletu gammou přepočítává, i když odpovídá hodnotě 1,0 a tudíž nedojde k žádné změně. Při barvení proběhne smyčka, který vstupní obraz pixel po pixelu

zpracuje a vykreslí výstupní obrázek vybranou paletou s aplikovanou gamma korekcí. Ve smyčce probíhá následující kód.

```
color = greyscaleInput.getRGB(xCounter, yCounter);
color = color & 0x000000ff;
drawSurface.setColor(new Color(
palettGamma[0][color],palettGamma[1][color],palettGamma[2][color]));
drawSurface.drawLine(xCounter, yCounter, xCounter, yCounter);
```

Funkce getRGB ze vstupního obrázku získá hodnotu jasu ve formě barvy. Druhý řádek kódu z ní separuje modrou složku. V tomto případě je úplně jedno, kterou separujeme, neboť šedotónový obraz je má všechny stejné. Následuje vykreslení nového pixelu barvou získanou z palety palettGamma.

5 APLET HUFFMANOVA KÓDOVÁNÍ

5.1 Teorie

5.1.1 Prefixový kód

Huffmanův kód je jeden z nejznámějších zástupců prefixových kódů, protože je také nejúčinnější. Vlastnost prefixového kódu je, že žádné kódové slovo není předponou (prefixem) jiného, takže je možné slova ihned po přijetí jednoznačně rozlišit [9]. Snižují se tím nároky na vstupní buffer dekodéru, jehož velikost musí být minimálně stejně velká, jako nejdelší kódové slovo.

5.1.2 Huffmanův kód

Algoritmus kódu definoval David Albert Huffman už v roce 1952. [9] Předpokladem zkrácení zprávy je nerovnoměrný výskyt obsažených znaků. Znaky časteji se opakující je možné reprezentovat kratší kódovou posloupností, než znaky nepravidelně se vyskytující. Pokud je výskyt znaků ve zprávě rovnoměrně rozložený, nedojde k požadovanému zkrácení.

5.1.3 Algoritmus sestavení kódu

1. Z množiny symbolů vytvoříme frontu a vybereme dva s nejnižší pravděpodobností výskytu (v případě apletu s nejnižší četností).
2. Tyto dva symboly sloučíme, vytvoříme nový a zařadíme ho do fronty. Jeho pravděpodobnost bude součtem pravděpodobností sečtených prvků.
3. Původním symbolům přiřadíme binární hodnoty 1 a 0. Nezáleží na tom, jak bude symbol s vyšší a nižší pravděpodobností označen, ale je nutné vybrané pravidlo dodržovat po celou dobu tvorby kódu. Pokud mají symboly stejnou pravděpodobnost, je jedno, jak je označíme. Zpracované symboly vyřadíme z fronty.
4. Frontu zpracováváme tak dlouho, dokud v ní nezbude pouze jeden prvek.
5. Nakonec projdeme vytvořeným stromem od kmene k větvím (symbolům) a každému symbolu přiřadíme výsledné kódové slovo podle hodnoty větve. [9]

5.1.4 Příklad Huffmanova kódu

Jako příklad si uvedeme zakódování textové zprávy „Hello, World“. Z prvního pohledu je patrné, že nejčastěji se opakují pouze dva znaky, takže kód bude mít vysokou

l	01
o	00
H	100
e	1011
,	1010
	1111
W	1110
r	1101
d	1100

Tab. 5.1: Kódovací tabulka příkladu „Hello, World“

entropii. Kódovací tabulku znázorňuje tabulka 5.1 a strom kódu může vypadat tak, jak prezentuje obrázek 5.2.

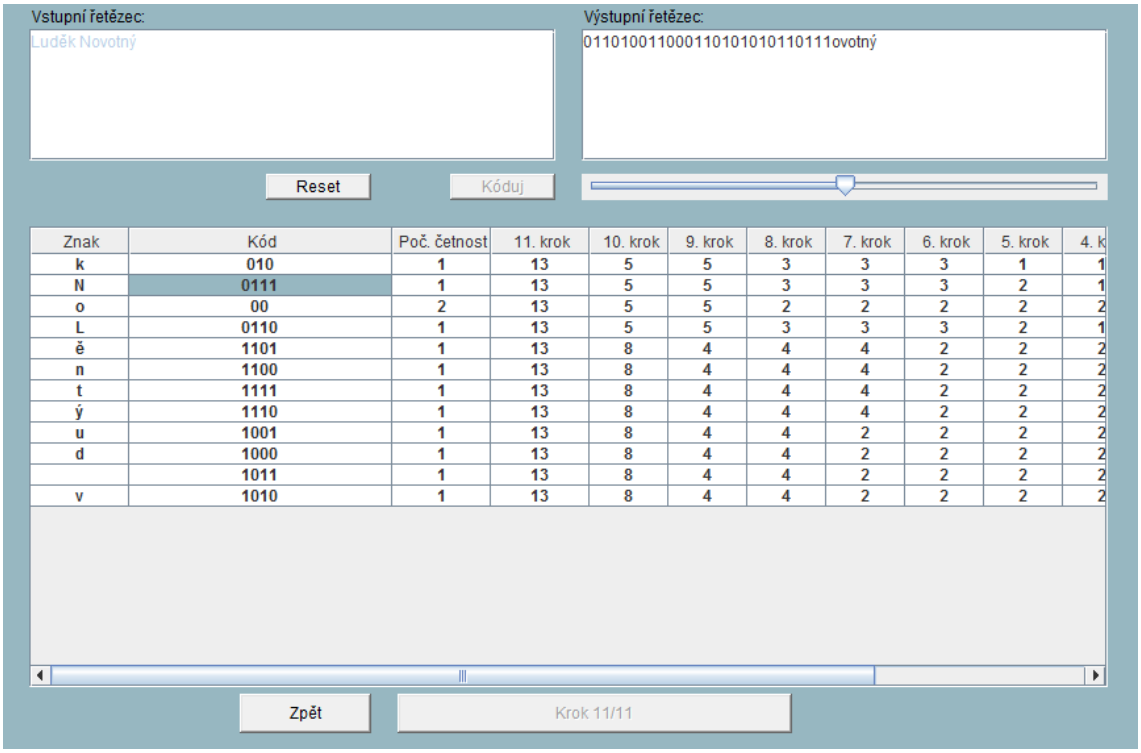
5.2 Popis apletu

Aplet je dle obrázku 5.1 tvořen dvěma poli pro vstupní a výstupní text, tabulkou pro tvorbu kódu a ovládacími prvky. Uživatel do levého vstupního pole vepíše text k zakódování a tlačítkem Kóduj odsouhlasí vložení. Následně klikáním na tlačítko Krok pod tabulkou provádí jednotlivé kroky algoritmu, konkrétně slučování prvků nejmenších četností. Tlačítko zároveň zobrazuje celkový počet kroků nutných k vytvoření kódu. Vrátit se o krok zpět umožňuje tlačítko Zpět.

Jakmile je kód vytvořen, zpřístupní se slider samotného kódování. Jeho tažením se jednotlivé symboly nahrazují příslušným kódovým slovem z tabulky. Zároveň se v tabulce barevně zvýrazňují nahrazované znaky. V jakémkoliv okamžiku je možno stisknout tlačítko Reset, které vyčistí tabulku a odemkne vstupní pole pro novou sekvenci znaků.

5.2.1 Tabulka kódu

Huffmanův kód můžeme přehledně znázornit grafem. Jelikož může uživatel zadat do vstupu jakkoliv dlouhý text, mohl by být graf až příliš veliký a nevlézt se do okna. Rozměrově úspornější je znázornění tabulkou, kdy každá řada reprezentuje jeden unikátní symbol a sloupce pak jejich četnosti a kódová slova. V každém kroku se pak vytvoří nový sloupec, který zobrazuje aktuální hodnotu četností. Pro názornost totiž všechny symboly zůstávají po celou dobu tvorby kódu v tabulce. Slučované symboly se také barevně zvýrazňují.



Obr. 5.1: Okno apletu Huffmanova kódování

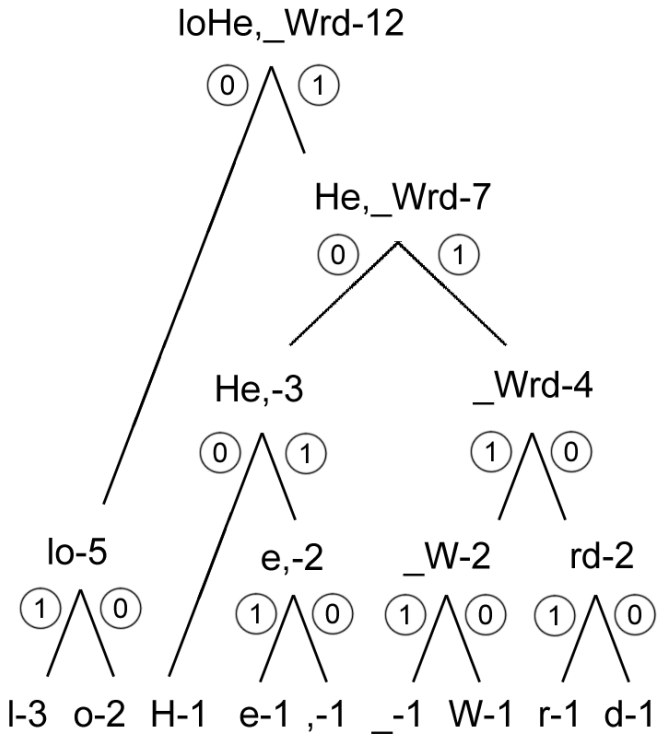
5.3 Interní funkce apletu

Použitý algoritmus programu přesně odpovídá teoretickému sestavení kódu. Po odsouhlasení vloženého textu tlačítkem Kóduj se vstupní text přesune do proměnné inputTextS typu String a zavolá se funkce countChar(), která každý znak přesune do instance třídy HashMap jménem characterDB. Klíčem je znak a hodnotou četnost. Pokud se znak už v hashovací tabulce nachází, zvýší se pouze jeho četnost. Nakonec se obsah tabulky characterDB vypíše do tabulky v okně apletu. Poté se jednotlivé symboly přesunou do prvku parents. Jedná se o instanci třídy List, přičemž tento seznam obsahuje pole integerů třídy ArrayList. V parents se následně vykonává fronta.

5.3.1 Fronta

Každý prvek v seznamu parents má následující strukturu dat. Na prvním místě pole se nachází četnost tohoto prvku. Další místa pole obývají čísla řádků symbolů, které tento prvek sdružuje. Parents tedy obsahuje jak unikátní symboly, tak i nové, vzniknuvší sloučením těchto symbolů.

Při každém stlačení tlačítka Krok se dvakrát spustí hledání prvku s nejmenší četností.



Obr. 5.2: Příklad Huffmanova kódu

```
for(int i=0; i<parents.size(); i++){
  if(parents.get(i).get(0) <= lastMin){lastMin = parents.get(i).get(0);
  lastMinRow = i;}
}
for(int i=0; i<parents.size(); i++){
  if((parents.get(i).get(0) <= lastMin2)&&(i != lastMinRow)){
  lastMin2 = parents.get(i).get(0); lastMinRow2 = i;}
}
```

Proměnná lastMin obsahuje četnost a lastMinRow řádek prvku v seznamu parents. Následně se vytvoří nový prvek, jehož četnost je součtem dvou nejmenších četností. Do nového prvku se také zkopírují řádky symbolů, které prvek sdružuje.

```
parents.add(new ArrayList<Integer>());
parents.get(parents.size()-1).add(lastMin+lastMin2);
for(int a=1; a<parents.get(lastMinRow).size() ;a++){
  parents.get(parents.size()-1).add(parents.get(lastMinRow).get(a));
}
for(int a=1; a<parents.get(lastMinRow2).size() ;a++){
  parents.get(parents.size()-1).add(parents.get(lastMinRow2).get(a));
}
```

Všem sdruženým symbolům prvku se adekvátně přiřadí hodnoty 1 a 0. Následně se zpracované prvky z fronty odstraní. Jelikož smazání jednoho prvku posune celý index seznamu parents, je nutné nejprve vymazat prvek na vyšším řádku.

```
if (lastMinRow>lastMinRow2){parents.remove(lastMinRow);
   parents.remove(lastMinRow2);}
else {parents.remove(lastMinRow2); parents.remove(lastMinRow);}
```

Nakonec se provede barvení zpracovaných buněk. Fronta se zpracovává tak dlouho, dokud v ní nezbude jediný prvek, který sdružuje všechny symboly. V grafu by se jednalo o kořen stromu.

5.3.2 Barevné zvýraznění buňek tabulky

Tabulka kódu v okně apletu je instancí třídy JTable, která bohužel neobsahuje přímé příkazy pro barvení určitých buněk. Bylo tedy nutné vytvořit novou třídu MyCellRenderer a přepsat standardní vykreslovací funkci getTableCellRendererComponent. Ta pracuje s indexem řádku a sloupce. Aby bylo vůbec možné vybarvit několik určitých buněk, je v této funkci vytvořena instance třídy ArrayList s názvem cellColor, tedy další pole, tentokrát stringů. Do něj se jako text ukládají po sobě jdoucí čísla. První značí index řádku a druhé index sloupce. Následující kód pak porovnává vložené hodnoty s aktuální hodnotou buňky a podle toho ji barevně zvýrazní.

```
if (cellColor.contains(String.format("%d%d", row,column)))
   {setBackground(new Color(151,183,193));}
else {setBackground(Color.WHITE);
}
```

6 ZÁVĚR

Cílem bakalářské práce bylo vytvořit programy pro interaktivní podporu výuky v kurzech BZSG, MGMP a BASS. Aplety byly napsány v jazyku Java, neboť mohou být bez větších potíží spouštěny přímo v internetovém prohlížeči nezávisle na operačním systému. Při psaní kódu byly využity dvě externí knihovny, JTransforms a JSyn.

První kapitola práce popisuje druhy signálů, věnuje se teorii fázorů a zmiňuje Fourierovu řadu. Následně je popsáno okno naprogramovaného apletu a jsou vyjmenovány jeho ovládací prvky. Nakonec je pospsáno, jak je sčítání fázorů v kódu realizováno skrze smyčku vykreslující řadu fázorů. Bylo též nutno zamezit problikávání prvků v okně využitím obrazového bufferu, což zmiňuje poslední odstavec kapitoly.

Druhá kapitola rozebírá téma Fourierovy transformace a kmitočtové filtrace. Jsou posány dva základní druhy transformací, a to DFT a FFT. Je popsáno okno apletu a jeho ovládací rozhraní. Poslední podkapitola rozebírá, jak jsou některé vstupní signály načítány z wav souborů a některé generovány, jak je v kódu implementovaná FFT skrze knihovnu JTransforms, jak se vzorky přehrávají díky knihovně JSyn a jak je realizována samotná kmitočtová filtrace nulováním potlačených modulů frekvenční modulové charakteristiky.

Druhá kapitola vyjmenovává několik metod barvení šedotónových obrazů, a to metodu parametrické křivky, metodu Schrödingerovy krychle a metodu barevné palety, kterou aplet využívá. Dále je popsáno okno a ovládací rozhraní apletu. Následně je popsáno, jak je v kódu implementované barvení metodou barevné palety a jak se na výstupní obraz aplikuje gamma korekce.

Čtvrtá kapitola se věnuje teorii Huffmanova kódu a je uveden algoritmus sestavení kódu. Je popsáno okno a ovládací rozhraní apletu. Následně jsou popsány základní části kódu, které se starají o realizaci algoritmu a je zmíněno, jakým způsobem bylo nutné programu vnutit funkci pro podbarvování buněk tabulky JTable.

LITERATURA

[1] SCHILDT, H. *Java: The Complete Reference.* 7. vyd. New York: McGraw-Hill, 2007. 1057 s. ISBN 978-0-07-163177-8.

[2] Eck, D.J. Introduction to Programming Using Java [online]. 2011, poslední revize 5.2013 [cit. 19.12.2013]. Dostupné na internetu: <http://math.hws.edu/eck/cs124/javanotes6/>

[3] SMÉKAL, Z. *Analýza signálů a soustav.* 1. vyd. Brno: FEKT VUT v Brně, 2012. 252 s. ISBN 978-80-214-4453-9.

[4] BIOLEK, D.; HÁJEK, K.; KRTIČKA, A.; ZAPLATÍLEK , K.; DOŇAR B. *Elektronické obvody I.* 1. vyd. Brno: Univerzita obrany, 2006. 318 s. ISBN 978-80-214-4453-9.

[5] BOLEČEK, L. Program pro zobrazení černobílých snímků v nepravých barvách. In *Elektrorevue* [online]. 2010 [cit. 20.12.2013]. Dostupné na internetu: <http://www.elektrorevue.cz/cz/clanky/zpracovani-signalu/15/program-pro-zobrazeni-cernobilych-snimku-v-nepravych-barvach/>

[6] ŠILHAVÝ, P. *Datová komunikace.* 1. vyd. Brno: Vysoké učení technické v Brně, 2011. 211 s. ISBN 978-80-214-4455-3.

[7] RAJMIC, P. *Základy počítačové sazby a grafiky.* 1. vyd. Brno: FEKT VUT v Brně, 2012. 161 s. ISBN 978-80-214-4451-5.

[8] HORÁK, D. Diskrétní transformace [online]. 2012 [cit. 20.05.2014]. 89 s. Dostupné na internetu: <http://mi21.vsb.cz/sites/mi21.vsb.cz/files/unit/diskretni_transformace.pdf>

[9] VLČEK, K. *Komprese a kódová zabezpečení v multimediálních komunikacích.* 2. vyd. Praha: Ben - technická literatura, 2004. 259 s. ISBN 80-7300-134-9.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

CPU *Central Processing Unit*

DFT *Diskrétní Fourierova transformace*

DP *Dolní propust*

FT *Fourierova transformace*

FFT *Rychlá Fourierova transformace*

HP *Horní propust*

JVM *Java Virtual Machine*

PP *Pásmová propust*

PZ *Pásmová zádrž*

A OBSAH ELEKTRONICKÉ PŘÍLOHY

Na přiloženém CD se nachází elektronická verze této bakalářské práce ve formátu PDF, zdrojové kódy apletů a balíky jar s příslušnými html soubory pro spuštění apletů offline. Struktura CD je následující:

- BakalarskaPrace.pdf
- aplet - složka s baliky jar a spouštěcími html soubory
  - fazory.html – otevře v prohlížeči aplet fázorů (offline)
  - barvy.html – otevře v prohlížeči aplet umělého barvení (offline)
  - huffman.html – otevře v prohlížeči aplet Huffmanova kódování (offline)
  - filtr.html – otevře v prohlížeči aplet kmitočtové filtrace (offline)
  - fazor.jar – balík zkompilovaného apletu fázorů
  - falsecolor.jar – balík zkompilovaného apletu umělého barvení
  - huffman.jar – balík zkompilovaného apletu Huffmanova kódování
  - soundf.jar – balík zkompilovaného apletu kmitočtové filtrace
- zdroj - složka se zdrojovými kódy
  - Fazory.java – zdrojový kód apletu fázorů
  - FlsClr.java – zdrojový kód apletu umělého barvení
  - Huffman.java – zdrojový kód apletu Huffmanova kódování
  - MyCellRenderer.java – zdrojový kód třídy náležející apletu Huffmanova kódování
  - SoundF.java – zdrojový kód apletu kmitočtové filtrace

Aplety jsou též dočasně umístěny na internetu pod následujícími odkazy:

- `http://indeegames.cz/wp-content/JavaApps/fazory.html`
- `http://indeegames.cz/wp-content/JavaApps/barvy.html`
- `http://indeegames.cz/wp-content/JavaApps/huffman.html`
- `http://indeegames.cz/wp-content/JavaApps/filtr.html`

Později budou přesunuty do společného seznamu apletů na odkazu: `http://www.utko.feec.vutbr.cz/~rajmic/applets/`.