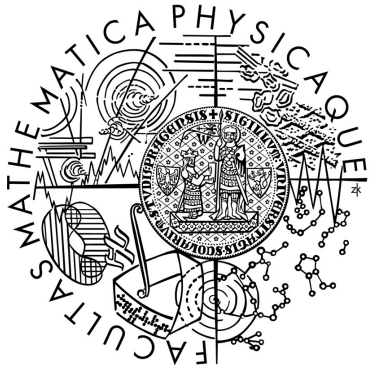
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Dávid Štrbka

Analýza hudobnej harmónie

Kabinet software a výuky informatiky

Vedoucí bakalářské práce: RNDr. Tomáš Holan, Ph.D.

Studijní program: Informatika
Studijní obor: Obecná informatika

Praha 2014

Poďakovanie

Rád by som poďakoval RNDr. Tomášovi Holanovi, Ph.D. za umožnenie vypracovania tejto práce a ochotu túto prácu konzultovať. Tiež by som sa chcel poďakovať rodine, ktorá ma podporovala. V neposlednej rade sa chcem poďakovať všetkým, ktorí mi pri tvorbe tejto práce pomohli.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V …..... dne.....

Název práce: Analýza hudobnej harmónie

Autor: Dávid Štrbka

Katedra / Ústav: Kabinet software a výuky informatiky

Vedoucí bakalářské práce: RNDr. Tomáš Holan, Ph.D.

Abstrakt: Hudobné skladby sú v dnešnej dobe bežnou súčasťou života. Hudobnou harmóniou definujeme jednu zo základných vlastností skladby a je to jeden z nosných subjektov zvukovej teórie. Na základe znalosti hudobnej harmónie, môžeme k danej skladbe reprodukovať hodobný doprovod. V úvode práca oboznámi čitateľa s teoretickými východiskami a možnými návrhmi algoritmov. Ďalšou súčasťou práce je implementácia programu, ktorý umožní automatickú analýzu hudobnej hamónie v zvukovom súbore. Program ponúka aj grafickú vizualizáciu zvukového spektra a harmónie a tiež po dodaní referenčných dát otestovanie úspešnosti analýzy. Na konci práce je zhodnotenie úspešnosti detekcie implementovaného programu.

Klíčová slova: hudba, harmonie, Fourierova transformace

Title: Analysis of Musical Harmony

Author: Dávid Štrbka

Department: Department of Software and Computer Science Education

Supervisor: RNDr. Tomáš Holan, Ph.D.

Abstract: Musical songs are ordinary part of our lives. Musical harmony is one of basic property for describing songs and also it ist large part of musical theory. By knowing musical harmony of some song, we can easily reproduce musical accompaniment. In introduction this thesis explain neccesary theory and possible ways of solving this problem. Next par is implementation of application for automatic chord recognition. Application provides graphical visualizations of sund spectrum and chord recognition data. After importing referention values application provides comparison of analysis efficiency. At the end of thesis is conclusion about efficiency of application detection algorithm.

Keywords: music, harmony, Fourier transform

Obsah

1 Úvod

V súčastnosti je hudba bežnou súčasťou našich životov. Či už vo forme hudobného priemyslu a pasívnym počúvaním jeho produktov, alebo aktívnou reprodukciou skladieb hraním na hudobných nástrojoch. Hudobná výchova patrí medzi povinné predmety vyučované na základných školách, kde sa prvý krát žiaci oboznamujú so základmi hudobnej teórie, kde sa stretneme aj s pojmom hudobnej harmónie.

1.1 Zvuk ako fyzikálny jav

Zvuk je mechanické vlnenie v látkovom prostredí. My budeme uvažovať o zvuku, ktorý vyvoláva vnem v ľudskom uchu, a teda ho môžeme počuť. Taký zvuk je vlnenie vo frekvenčnom rozsahu 20 Hz až 20kHz. Ďalej môžeme zvuk rozdeliť medzi hudobný a nehudobný, kde je rozdiel ten, že nehudobný zvuk má nepravidlné kmitanie, tento rozdiel je ale značne subjektívny a záleži od individuálneho vnímania jednotlivca. Hudobný zvuk nazývame tón a nehudobný je hluk.

Okrem frekvencie tohto vlnenia je dôležitý parameter intenzita zvuku, čo je vlastne amplitúda kmitov. Intenzitu zvuku najbežnejšie meriame v jednotke dB, čo je bezrozmerná jednotka definovaná ako desaťnásobok dekadického logaritmu pomeru dvoch výkonov, alebo intenzít.

1.2 Základy hudobnej teórie

Hudobná teória je časť muzikológie zahrnujúca hudobnoteoretické disciplíny, ktoré sa zameriavajú na výskum synchronických aspektov hudby alebo hudobnej kultúry[1]. Na to má svoje termíny a metodológiu na popis zvuku, ktoré umožňujú abstrahovať od zvuku ako fyzikálneho javu. Tieto pojmy si popíšeme, lebo sa vyskytujú ďalej v tejto práci.

Tón – je každý zvuk so stálou frekvenciu. Je to základný stavebný kameň hudby. Tón je popísaný základnými vlastnosťami: výškou, dĺžkou, silou a farbou. Výška tónu je určená jeho frekvenciou. Dĺžka je určená časom znenia tónu. Sila je určená amplitúdou tónu. Farba je určená zdrojom tónu. Fyzikálne ju môžeme určiť na základe frekvencií súčasne znejúcich s dominantnou frekvenciu tónu. Dominantná frekvencia je tá, ktorú najviac počujeme a je vždy násobkom základnej frekvencie. Spolu s ňou však znejú aj tzv. harmonické frekvencie, ktoré sú tiež násobkom základnej frekvencie. Tieto frekvencie spolu určújú farbu tónu respektíve zvuku.

Ľudské ucho nevníma absolútny rozdiel frekvencií medzi dvoma tónami, ale ich pomer. V praxi je určená len frekvencia jedného tónu a ďalšie sú od neho dovodené. Týmto tónom je tón nazývaný komorné "a" s frekvenciou 440 Hz. Existuje 7 základných tónov c, d, e, f, g, a, h. Najmenšia vzdialenosť medzi tónami je definovaná ako poltón. Medzi niektorými tónami je vzdialenosť dva poltóny, medzi inými len jeden poltón.

Tón s dvojnásobnou frekvenciou nejakého tónu je rovnaký tón, len takzvane o oktávu vyšší. Oktáva je uzavretý tónový interval keď je vzdialenosť dvomi tónami 7 základných tónov, respektíve 11 poltónov.
Hudobný prejav je definovaný týmito základnými prvkami: melódiou, harmóniou a rytmusom. Rytmus je časová zložka hudby, ktorá vyjadruje vnútorné členenie metrických jednotiek, striedanie rôznych dĺžok tónov. Melódia vyjadruje sekvenciu tónov hrajúcich za sebou takzvaný hudobný motív.

1.2.1 Hudobná harmónia

Je prvok hudobného prejavu, ktorý je záujmom tejto práce. Samotný pojem označuje súzvuk. V širšom zmysle je to náuka o akordoch a tvorbe sekvencií akordov.

Akord je súzvuk troch, alebo viacerých tónov. Akordov je veľa typov, ale vyjadrené sú vždy základným tónom a typom. Typ akordu je určený hudobnými intervalmi medzi základným tónom a ostatnými tónami v akorde. Pre ilustráciu základný a najčastejší typ akordu je durový kvintakord, ktorý sa niekdy ani neoznačuje samostatnou značkou. Tento akord pozostáva zo základného tónu, veľkej tercie a kvinty, čo sú hudobné intervaly medzi základným tónom a ostatnými dvomi tónami v akorde. Napríklad Cdur obsahuje tóny (C,E,G).

1.3 Cieľ práce

Cieľom práce bude vytvorenie aplikácie na analýzu hudobnej harmónie v skladbách. Pod tým myslíme rozdelenie skladby na úseky v ktorých je rovnaká hudobná harmónia. Túto harmóniu budeme označovať harmonickými značkami. V nasledujúcich kapitolách sa budeme venovať spracovaniu digitálneho audio signálu. Oboznámime s možnosťami riešenia tohto problému na základe vykonaných výskumov v tejto oblasti. Rozoberieme návrh algoritmu a jeho implementáciu v našej aplikácií. Aplikácia bude naprogramovaná v jazyku C# na platforme OS Windows. Bude mať GUI, ktoré bude poskytovať informácie v grafickej, ale aj textovej podobe. Textová podoba umožní prípadné ďalšie strojové spracovanie. Aplikácia umožní porovnanie dvoch textových výstupov a tiež bude mať základné ovládacie prvky na prehrávanie analyzovanej skladby. Na konci zhodnotíme úspešnosť našej aplikácie.

1.4 Štruktúra práce

V tejto kapitole sme rozobrali cieľ práce a nevyhnutý úvod do problematiky.

V kapitole 2 sa rozoberá analýza z hľadiska práce s digitálnym zvukom. Ide tiež o akýsi úvod do problematiky.

V kapitole 3 sa rieši analýza cieľového problému tejto práce a na konci náčrt riešenia.

V kapitole 4 sa podrobne preberá implementácia.

V kapitole 5 je užívateľská dokumentácia.

V kapitole 6 je zhodnotenie aplikácie, porovnanie s implementáciami a záver.

V kapitole 7 je zoznam použitej literatúry.

2 Analýza problému – spracovanie zvuku v PC

V tejto kapitole rozoberieme problém analýzy hudobnej harmónie z technického aj algoritmického hľadiska ako sa dopracovať k výsledku.

Na začiatok si predstavme jednotlivé hlavné body, ktoré treba vyriešiť.

1. Na ďalšie spracovanie audiosignálu v PC, je potrebné vyriešiť digitálne spracovanie audiosignálu, aby sme na ňom mohli robiť výpočty.
2. Návrh samotného algoritmu.

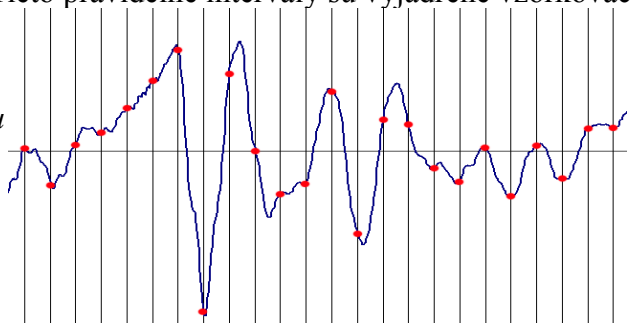
2.1 Úvod do digitálneho záznamu audiosignálu

Predpokladom spracovania zvuku na počítači je jeho prevedenie do digitálnej podoby. Aby sme pochopili neskoršie algoritmy, potrebujeme poznať základy digitalizácie zvukového signálu a jeho reprezentácie v digitálnej podobe.

Digitalizácia zvukového signálu prebieha v dvoch krokoch. Najprv sa spojitý signál musí navzorkovať. To znamená, že v pravidelných intervaloch sa odčíta hodnota signálu. Tieto pravidelné intervaly sú vyjadrené vzorkovacou frekvenciou.

Obrázok 1: Vzorkovanie zvukového signálu

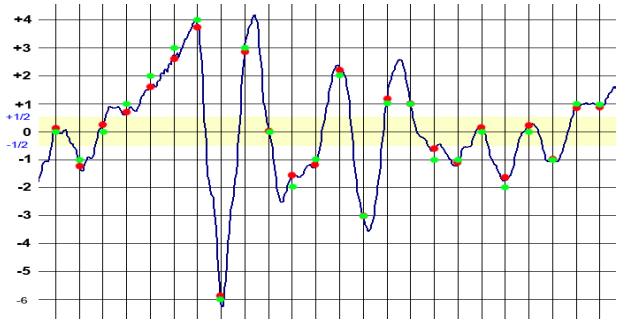
Prevzaté z [5]



Keďže hodnota daných vzorkov reprezentuje reálne číslo, ktoré v počítači nevieme spracovávať presne, musíme daným vzorkom priradiť hodnotu s nejakou presnosťou. Tento proces sa volá kvantovanie. Rozlíšenie kvantovania vyjadrujeme v bitoch, ktoré určujú rozsah hodnôt, ktorý vzorok môže nadobúdať.

Obrázok 2: Kvantovanie zvukového signálu

Prevzaté z [5]



Tento proces digitalizácie, využíva Pulzne-kódová modulácia (PCM), čo je spôsob kódovania najčastejšie používaný na digitálny záznam zvukového signálu.

Špeciálnym podtypom je lineárna PCM (LPCM), kde sú úrovne kvantovania lineárne odstupňované. LPCM je štandardné kódovanie na Audio CD v parametroch: vzorkovacia frekvencia 44100kHz a kvantizačné rozlíšenie 16-bit.

2.2 Digitálne zvukové formáty

Digitálny zvukový záznam sa v počítači ukladá v rôznych zvukových formátoch. Najbežnejším formátom bez kompresie umožňujúcim ukladať zvuk v kódovaní LPCM je formát Waveform audio file format (WAV). Samotný formát WAV zodpovedá norme RIFF.

Existujú aj formáty s kompresiou, bezstratovou aj stratovou, najpoužívanejší v súčastnosti je stratový formát MP3. Podpora zvukových formátov v aplikáciach je zásadná vec z hľadiska používateľa. Preto treba zvážiť pri aplikácií, čo najväčšiu podporu formátov, lebo absencia podpory môže odradiť časť používateľov od samotnej aplikácie, aj pri dosahovaní dobrých výsledkov.

Z programátorského hľadiska to pridáva prácu a jedna z možností je použiť už naprogramovanú knižnicu na prácu so zvukom, ktorá podporuje viaceré formáty. To je aj prípad našej aplikácie, ktorá používa knižnicu NAudio a tak podporuje formáty WAV aj MP3, čo sú najpoužívanejšie formáty na záznam zvuku pre bežných užívateľov.

3 Analýza problému z algoritmického hľadiska

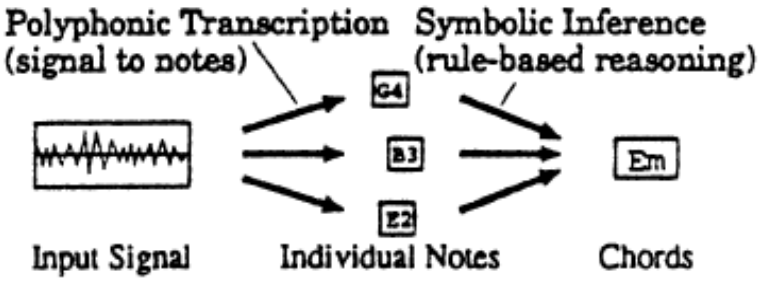
V tejto kapitole detailnejšie preberieme problém z hľadiska návrhu algoritmu. V prvej podkapitole načrtneme možné riešenie ako celkový algoritmus. V nasledujúcich podkapitolách preberieme čiastkové algoritmy a rôzne modifikácie a zlepšenia. V poslednej kapitole zhrnieme návrh algoritmu našej aplikácie.

3.1 Možnosti riešenia

Ako sme sa už skôr dozvedeli, hudobná harmónia sa zaoberá súčasným znením viacerých tónov, či už vo forme akordov, alebo všeobecne. Ideálnym prípadom by bolo, keby sme zistili priamo notový zápis danej skladby. Z toho by sme vedeli jednoznačne určiť danú harmóniu. To je vlastne základ intuitívneho algoritmu:

1. Zistiť jednotlivé tóny, ktoré hrajú a kedy hrajú.
2. Z toho vypočítať harmóniu.

Prvý bod tohto algoritmu predstavuje problém prepisu polyfonickej hudby, ktorým sa zaberali v článkoch [6] a [7]. Jednalo sa prvé práce zaoberajúce sa týmto problémom. Rozpoznávanie akordov pomocou tejto metódy je spravené v [8]. Ako je spomenuté v [2] tieto systémy pracujú na základe prepisu hudby na noty a potom použitia nejakého expertného systému na odvodenie hraných akordov.



Obrázok 3: Architektúra tradičného intuitívneho algoritmu

Prevzaté z [2]

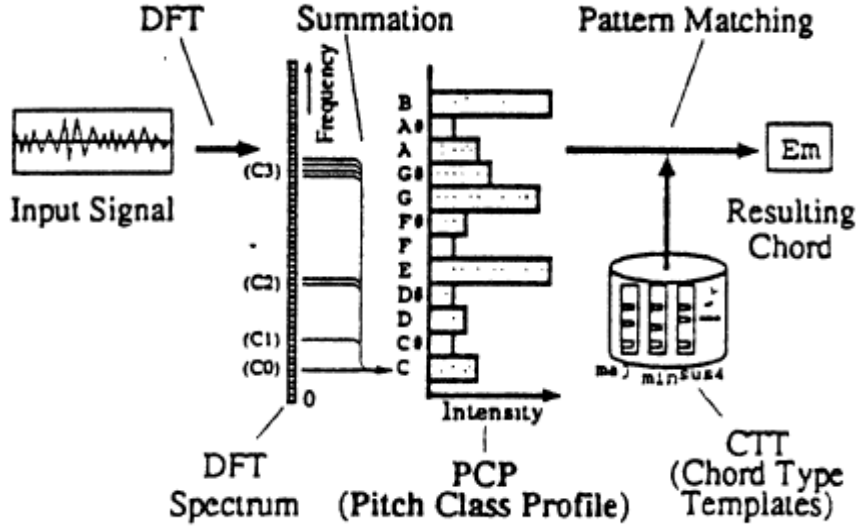
Problémom je, že úloha samotného prepisu polyfonickej hudby na noty je veľmi zložitá a pri tomto procese môže vzniknúť veľa chýb.

Ďalej sa týmto problémom zaoberá *Realtime chord recognition of musical sound: A system using common lisp music.* [2], kde prišli s novým návrhom algoritmu. Spomenutý algoritmus zaviedol ideu použitia tzv. Pitch class profile (PCP), v mojom preklade poltónových profilov. Tento návrh má výhodu vo väčšom využití informácie, lebo akordy sa určujú nie na základe jednotlivých vyextrahovaných nôt zo signálu, ale z celého spektra.

Vysvetlime si teraz bližšie tento návrh algoritmu na Obrázku 4, ktorý pozostáva s nasledujúcich krokov:

1. Najprv sa signál prevedie z časovej do frekvenčnej oblasti pomocou diskrétnej Fouriérovej transformácie.
2. Z výsledného spektra sa vytvorí poltónový profil
3. K danému profilu sa priradí akord.

Ako vstup sa myslí krátky úsek, rádovo v ms, zvukového signálu. Ozrejmíme si bližšie princípy krokov 2 a 3.



Obrázok 4: Architektúra algoritmu

Prevzaté z [2]

Väčšina ďalších článkov zaoberajúcich sa týmto problémom je postavená na základnom návrhu tohoto algoritmu.

3.1.1 Pitch class profile(PCP)

PCP profil je vlastne vektor 12 základných poltónov vyjadrujúci ich intenzity. Základná idea je tá, že rovnaké poltóny zahraté v iných oktávach sú z hľadiska hudobnej harmónie rovnaké. Takže nepotrebujeme poznať každý jednotlivý hraný tón v zvukovej skladbe. Matematické vyjadrenie poltónového profilu je

$$PCP(p)=\sum_{k:M(k)=p}\left|X(k)^2\right|$$

a

$$M(k)=\lfloor 12\log_2(k/N*f_{sr}/f_{ref})\rfloor \bmod 12$$

kde X je lineárne frekvenčné spektrum získané Fouriérovou transformáciou, f_{sr} je vzorkovacia frekvencia a f_{ref} je referenčná frekvencia prvého poltónu v PCP. N je celkový počet dielov spektra FT.

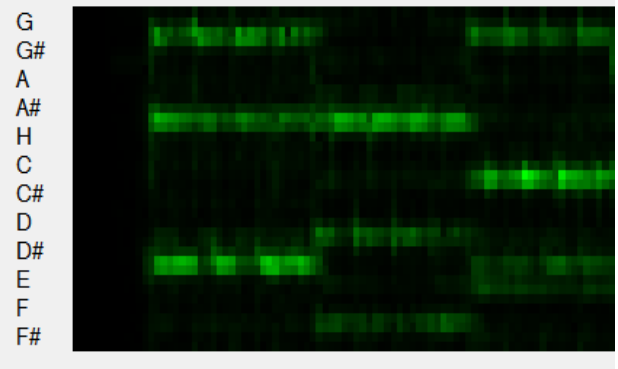
Ak by sme chceli spracovávať spektrum logaritmicky rozdelené získané Konštantnou Q transformáciou, tak matematické vyjadrenie by bolo

$$PCP(p)=\sum_{m=0}^{M-1}\left|X_{CQ}(p+12\mathrm{m})\right|$$

kde M je počet oktáv v spektre z CQ transformácie.

Samotnú ideu PCP profilu môžme rozšíriť, že nepôjde o 12 miestny vektor, ale bude mať veľkosť násobku čísla 12. S tým, že potom dôjde k výslednému zmenšeniu na 12 miest, alebo sa budú robiť aj ďalšie operácie na napr. 24 miestnom PCP profile.

Spomeňme ešte, že pole PCP profilov ktoré zasebou nasledujú v skladbe tvorí tzv. chromagram. Niekedy sa tieto názvy zvyknú zamieňať a pod chromagramom môžeme mať namysli PCP profil. Čisto jazykovo vzaté chromagram je až samotné vykreslenie týchto dát ako na Obrázku 5.



Obrázok 5: Chromagram

3.1.2 Priraďovanie akordov k PCP profilom

Ak už máme vytvorený chromagram, resp poznáme jednotlivé PCP profily, tak k nim môžeme priraďovať akordy. Ide o problém tzv. pattern matching. Najprv si vytvoríme vzorové PCP profily, ideálnych akordov. Napríklad profil akordu Edur(G#, H, D#) bude (0,1,0,0,1,0,0,0,1,0,0,0) s tým, že prvé prvok je tón G prvky s vyším indexom sú vyššie.

Potom sa budeme snažiť k rozpoznávanému profilu priraďovať vzorový profil.

Na to existuje veľa metód, v [2] sú spomenuté nasledovné:

1. Druhá mocnina euklidovskej vzdialenosti, vyberáme najmenšie skóre pre
$$Score(p)=\sum_{p=0}^{11}(T_c(p)-PCP(p))^2$$
2. Vážený skalárny súčin, vyberáme najväčšie skóre
$$Score(p)=\sum_{p=0}^{11}W_c(p)*PCP(p)$$
V obidvoch prípadoch vektory T_c a W_c , kde c označuje akord vychádzajú zo vzorových PCP profilov, ale môžu byť upravené.

Ja som skúšal použiť aj iné metriky napr. Hammingovu vzdialenosť, ale neprinieslo to želaný efekt.

3.2 Fouriérova transformácia a Hammingová oknová funkcia

Fouriérova transformácia (ďalej FT) je matematická metóda na prevod signálu z časovej do frekvenčnej oblasti. Definovaná na spojitom signáli je ako integrálna transformácia S(w) funkcie s(t).

$$S(w)=\int_{-\infty}^{\infty} s(t)e^{-iwt}dt$$

Keďže my pracujeme so signálom digitalizovaným, ktorý je vzorkovaný, bude nás bližšie záujimať jej diskrétna verzia. Diskrétna furiérova transformácia (ďalej DFT):

$$X_k=\sum_{n=0}^{N-1} x_n * e^{(-i2\pi kn/N)}$$

,kde $x_0 .. x_{N-1}$ je transformovaná sekvencia kompexných čísiel.

Pre bližšie zoznámenie sa s touto transformáciu a algoritmom na jej výpočet poslúžia skriptá [10].

Pre naše účely aplikujeme na vstupné dáta DFT Hammingovu oknovú funkciu, ktorá hodnoty v tzv. okne, prenásobí koeficientami, tak že dáta na okrajoch okna budú zoslabené. Prečo a ako sa používa je dobre objasnené v [12]. Tiež je tam objasnený princíp použitia tzv. overlap-u, opakovania časti vstupných dát DFT , ktorý tiež používame.

3.3 Algoritmus Constant Q transform

Algoritmus Constant Q transform je podobná transformácia ako Fouriérova, ale výstup je spektrum, ktoré je rozdelené logaritmicky.

Frekvencia k-tého dielu spektra je $f_k=(2^{(1/B)})^k * f_{min}$, kde B je počet dielov v jednej oktáve a f_{min} je minimálna frekvencia, ktorú si nastavíme rovnako ako aj maximálnu frekvenciu.

Výhodná je práve preto, že keď si B zvolíme 12, tak jednotlivé diely spektra predstavujú poltóny.

Q predstavuje tzv. Quality faktor a je to konštanta. Nastavuje sa ako vstupný parameter tejto tranformácie. Definovaná je ako

$$Q=f_k/delta\ f_k=f_k/(f_{(k+1)}-f_k)=(2^{(1/b)}-1)^{-1}$$

Bližšie zoznámenie sa s touto transformáciu a algoritmom na jej výpočet nájdete v [4]

3.4 Možnosti zlepšenia priraďovania akordu k PCP profilu

V zásade sa ponúkajú 2 cesty:

1. Proces tvorby výsledného PCP profilu
2. Proces priraďovania akordu k PCP profilu (v ang. matching)

3.4.1 Proces tvorby PCP profilu

1. Mať na začiatku väčšie rozlíšenie spektra, resp. chromagram s väčším počtom dielov v jednej oktáve. Z neho vieme vygenerovať presnejší výsledný PCP profil(12 poltónov). Ak je samotný poltón ešte rozdelený na niekoľko dielov, napríklad 3, tak z hodnôt z týchto dielov vieme získať presnejšiu polohu poltónu vo výslednom profile. Stačí použiť napr. Lagrangeovu interpoláciu týchto bodov a z výsledného polynómu zobrať bod nachádzajúci sa v bode jeho nulovej derivácie.

2. Rôzne úpravy profilu, napríklad doľaďovanie, lebo vstupné nahrávky nie sú vždy dokonale vyladené, spomenuté v [13]. Ďalšou možnosťou je aplikovať vyhlaďovanie (z ang. smoothing), kedy sa časť z predchádzajúceho profilu prenáša do aktuálneho. Nevadí to, lebo hraný akord trvá zvyčajne dlhšiu dobu, ako jeden PCP profil, naopak zníži to šum. Spomenutá heuristická metóda sa používa v [2].

3.4.2 Proces priraďovania akordu

Do procesu samotného priraďovania akordu môžeme zapojiť aj metódy strojového učenia a umelej inteligencie, ktoré asistujú pri rozhodnutí aký akord použiť. V našej implementácií som ich nepoužil, ale spoločné majú to, že využívajú informácie jednak externé a potom informácie o priebehu rozpoznávania harmónie celého súboru a nie sú izolované na analyzovaný krátky časový úsek.

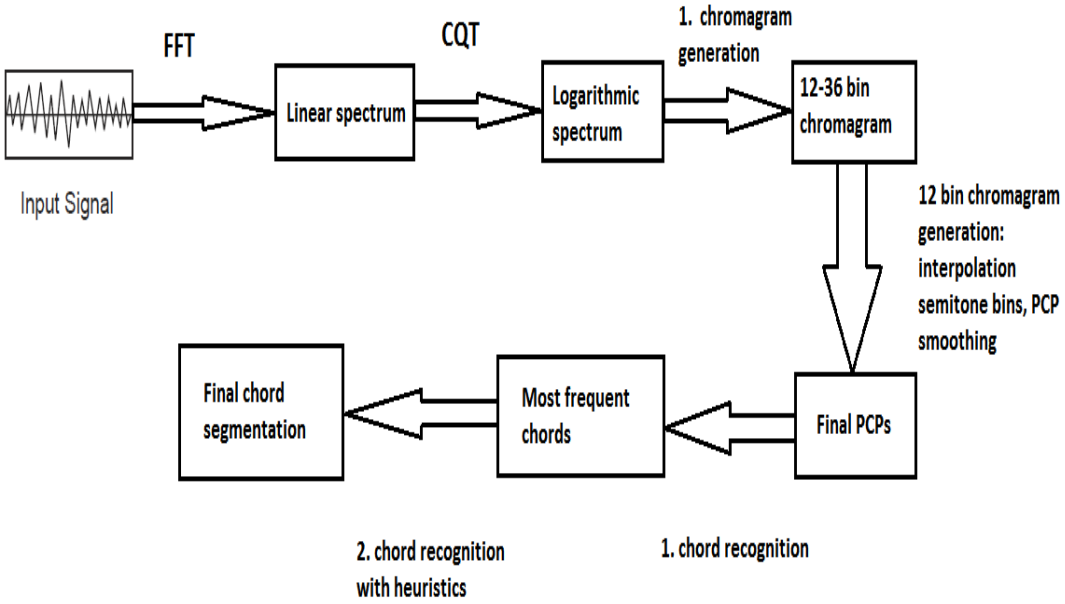
Jednou z tých metód je Hidden Markov Model a súvisiace algoritmy, čo nebudeme bližšie rozoberať. Pre ilustráciu princípu
, dopredu sa zadá harmónia na daných skladbách a daná metóda sa podľa toho naučí pravdepodobnosti, nasledovania konkrétneho akordu, po nejakej postupnosti akordov. Pri neznámej skladbe ich potom použije.

Ja som vymyslel jednu jednoduchú heuristiku, ktorá tiež využíva už vypočítané dáta, ktorú predstavím v mojom návrhu algoritmu.

3.5 Môj návrh algoritmu

Môj algoritmus vychádza zo základnej štruktúry na Obrázku 4. Detailne je rozobratý v implementácií, takže načrtnem len jeho schému a moju heuristickú metódu.

1. Pomocou FFT vygenerovanie lineárneho spektra.
2. Pomocou CQT jeho transformácia na spektrum logaritmicky odstupňované
3. Transformácia tohoto spektra na chromagram 12-36 dielový podľa zadaných parametrov
4. Tvorba 12 poltónových PCP profilov z chromagramu, aplikovanie interpolácie a smoothingu.
5. 1. detekcia akordov a z nich zistenie najpočetnejších.
6. 2. detekcia akordov, zobratie do úvahy najpočetnejšie akordy z 1. detekcie podľa parametrov.
7. Tvorba konečnej segmentácie.



Obrázok 6: Návrh algoritmu v mojej aplikácií

Daná heuristika ma napadla pri tej príležitosti, že väčšina populárnej hudby je harmonicky veľmi jednoduchá. Veľa piesní používa len 4 akordy. Zároveň môj algoritmus nemá až tak dobre ošetrené filtrovanie rôznych nehudobných zvukov, kde počítam aj spev, bicie nástroje a podobne, ktoré sa vyskytujú v pop music. Veľmi dobré výsledky dáva na inštrumentálnej hudbe, napr. len klavírny doprovod. Ak dopredu zistíme pri populárnej hudbe najčastejšie akordy, tak v druhom zisťovaní, môžeme zoslabiť pravdepodobnosť výskytu ostatných akordov, čo prinesie lepšie výsledky.

4 Implementácia aplikácie HarmonyAnalyser

V tejto kapitole rozoberieme návrh aplikácie z implementačného hľadiska a detaily implementácie aplikácie.

4.1 Všeobecná architektúra programu a platforma

Aplikácia je naprogramovaná v objektovo orientovanom jazyku C#, pod platformou Microsoft .NET 4.0 na operačnom systéme Microsoft Windows. Vývoj aj výsledné zostavenie aplikácie bolo realizované v prostredí Microsoft Visual Studio 2010. Na grafické užívateľské rozhranie je použité interné .NET API Windows Forms. Práca so zvukovými súbormi je zabezpečená open source knižnicou NAudio, podrobnejšie informácie o tejto knižnici nájdete na [http://naudio.codeplex.com/].

4.1.1 Hlavné funkčné bloky aplikácie

Prácu aplikácie môžeme rozdeliť do nasledujúcich logicky nezávislých funkčných blokov:

1. Hlavný proces aplikácie, GUI
2. Analýza
3. Prehrávanie analyzovaného súboru
4. Export, import a porovnávanie výsledkov analýzy

4.1.2 Základná funkcionalita aplikácie

Po spustení sa zobrazí hlavné okno s GUI. Užívateľ si najprv vyberie zvukový súbor na ktorom chce previesť analýzu. Potom spustí analýzu. Výsledok sa zobrazí v hlavnom okne. Ďalej má možnosť si daný súbor prehrať, s tým, že v GUI uvidí počas prehrávania zvukového súboru súvislosť s výsledkami analýzy. Ďalej má možnosť exportovať výsledky analýzy, prípadne importovať vzorové výsledky na porovnanie. Posledná možnosť je upraviť štandardne preddefinované parametre analýzy a urobiť novú analýzu, na rovnakom, alebo novo zvolenom súbore.

4.2 Hlavný proces aplikácie a GUI

Aplikácia je z hľadiska GUI tvorená jedným oknom *Form1* na ktorom sú všetky ovládacie prvky.

4.2.1 Trieda Form1

Form1 je trieda hlavného okna, ktorej inštancia sa vytvorí hneď po spustení programu zavolaním funkcie Main v statickej triede *Program.* Je to hlavná trieda na obsluhu grafického užívateľského rozhrania, preto si rozoberieme všetky metódy tejto triedy. Obsahuje tieto podstatné ovládacie prvky: menu menuStrip1 s tlačítkom Settings, talčítka buttonOpen, buttonAnalyse, buttonExport, buttonImport, buttonCompare, label1, pictureBox1-3, comboBox1, listBox1-2, waveformPainter1 na ktoré sa môžeme odkazovať v ďalších podkapitolách.

4.2.2 Metóda buttonOpen_click a metódy buttonPlay_click a button_Stop_click

Metóda buttonOpen_click slúži na zvolenie zvukového súboru s ktorým chceme pracovať. Realizované je to pomocou štandardného GUI prvku OpenFileDialog. Okrem vybratia úboru sa v tejto metóde vytvorí inštancia _player triedy AudioPlayer a parametrom je zvolený súbor. Druhá činnosť je nastavenie metódy SampleAggregator_MaximumCalculated, SampleAggregatoru v playeri, ktorá bude volaná pri udalosti MaximumCalculated.

Metódy buttonPlay_click a button_Stop_click slúžia na začatie a zastavenie prehrávania. Realizujú to zavolaním príslušných metód triedy AudioPlayer.

4.2.3 Metóda buttonAnalyse_click

Metóda buttonAnalyse_click slúži na zahájenie výpočtu analýzy a je spustená po stlačení tlačítka buttonAnalyse. Na začiatku si inicializuje niektoré premenné a zablokuje GUI objekty v hlavnom formulári, ktoré by sa nemali spúšťať počas analýzy.

Hlavná náplň tejto metódy je vytvorenie jednotne kódovaného dátového prúdu ms typu MemoryStream, ktorý obsahuje zvukové dáta. Cieľom je dostať jednokanálový(mono) a 16-bitovo kvantovaný PCM kódovaný formát. Realizované je to pomocou tried WaveFileReader, MP3FileReader, WaveStream a WaveConversionStream knižnice NAudio. Bližší popis čítania zvukových dát pomocou triedy WaveStream je popísaný v kapitole 4.4 Prehrávanie. Triedy WaveFileReader, MP3FileReader, technicky zabezpečujú čítanie zo zvukových súborov a sú ako parameter konštruktora triedy WaveStream. Parametrom ich konštruktora je samotná cesta k zvukovému súboru.

Prevod na jednokanálový súbor je zabezpečený prepísaním len polovice načítaných dát do výsledného prúdu ms.

Potom sa vytvorí inštancia _analyzer triedy MainAnalyser, ktorá dostane ako vstupné parametre konštruktora dátový prúd ms a jednolivé parametre uložené v inštancií triedy Settings _presets. Nastaví sa metóda UpdateProgress, ktorá je volaná pri akcií _analyzer.ProgressCallback.

V novom vlákne sa zavolá metóda _analyzer.PerformAnalysis, ktorá vykoná danú analýzu. Potom sa vytvorí inštancia triedy Visualizations a zavolajú sa jej funkcie na vizualizáciu dát vyporodukovaných pri analýze, ktoré sa uložia do premenných typu Bitmap. Menovite sú to spektrogram z Fouriérovej analýzy, pôvodný chromagram, výsledná segmentácia akordov, vyhladený výsledný 12 poltónový chromagram, a špičky z chromagramu. Prvé tri z nich sa potom nastavia GUI objektom PictureBox1-3 v hlavnom okne. Posledná časť je tvorba typu AnalysationDataGeneral z analyzovaných dát, ktoré sa zobrazia v listBox1.

Po skončení tohoto vlákna sa odblokujú zablokované GUI prvky.

4.2.4 Metóda SampleAggregator_MaximumCalculated

Metóda SampleAggregator_MaximumCalculated sa volá pri vyvolaní udalosti MaximumCalculated triedou SampleAggregator v triede AudioPlayer. Slúži na

riadené vykreslovanie údajov prehrávaného súboru, bližší popis je v kapitole venovanej triede SampleAggregator.

Zabezpečí pridanie maximálneho vzorku GUI objektu waveformPainter1, ktorý vykresluje vlnový priebeh prehrávaného súboru. Ďalej podľa aktuálneho času nastaví pozíciu objektu trackBarPosition, ktorí zobrazuje časový priebeh prehrávaného súboru.

Ďašia vec je zobrazenie aktuálneho času prehrávania a danej harmónie v objekte label1. Posledná vec je zavolanie metód UpdateVisualizations inštancií _updater a _updater2 triedy VisualizationsUpdater, ktorá zabezpčuje zobrazenie aktuálnej časovej stopy vo vizualizáciach v pictureBox1(segmenty harmónie) a pictureBox2(spektrogram).

Táto metóda sa volá vždy po prehratí 10 vzorkov.

4.2.5 Metódy buttonExport_click a buttonImport_click

Metódy buttonExport_click a buttonImport_click slúžia na import a export analyzovaných dát. Fungujú štandardným spôsobom použitím objektu OpenFileDialog resp SaveFileDialog.

Po vybratí súborov sa vykoná serializácia, resp deserializácia pomocou postupu popísaného v kapitole 4.5. Pri serializácií sa ukladajú dáta z prvku _analyzer a pri deserializácií sa dáta uložia do prvku AnalysationDataGeneral _imported_data, ktoré sa zároveň zobrazia v listBox1.

4.2.6 Metóda buttonCompare_click

Metóda buttonCompare_click slúži na porovnanie výsledku analýzy a importovaného výsledku. Skontroluje či sú obidva výsledky vytvorené a načítané, potom vytvorí inštanciu triedy AnalysationDatasComparer zavolá jeho metódu CompareWithGeneral a zobrazí výsledok v dialógovom okne.

4.2.7 Ostatné pomocné metódy triedy Form1

Metóda settingsToolStripMenuItem_Click vytvorí inštanciu FormSettings a zobrazí okno v modálnom móde. Slúži na zmenu parametrov analýzy uložených v _presets.

Metóda comboBox1_SelectedIndexChanged slúži na prepínanie zobrazených vizualizácií v pictureBox2.

4.2.8 Trieda Visualizations a jej metódy

Trieda Visualizations slúži na vykreslenie vygenerovaných dát z analýzy. Obsahuje metódy RenderSpectrogram, RenderChromagram, RenderChromaExtract, RenderSegmentation a GetColor.

Pomocná metóda GetColor slúži na výpočet intenzity farby k danej amplitúde. Na výpočet potrebuje ako vstupné parametre najmenšiu, najväčšiu a rozsah hodnôt spektra. Metóda je to univerzálna, takže pod spektrom sa myslí spektrum hodnôt.

Renderovacie metódy fungujú analogicky, takže popíšeme všeobecné fungovanie. Každá má ako parameter referenciu na pictureBox a dátový typ vykreslovaných dát.

Na začiatku sa zistia potrebné parametre pre metódu GetColor a rozmery referenciou predaného objektu typu pictureBox. Potom sa vytvorí objekt typu Bitmap canvas, v dvoch cykloch dĺžky rozmerov odkazovaného pictureBoxu sa canvas plní pixelmi s farbou určenou metódou GetColor. Parameter amplitúdy pre metódu GetColor sa počíta zistením indexu do dátového objektu. Tento index je vypočítaný škálovaním medzi rozmermi pictureBoxu a rozmermi dátového objektu, čo je zvyčajne objekt typu ArrayList. Na ilustráciu prikladám príslušný zdrojový kód metódy RenderChromagram

```
double amplitude = ((double[])_chromaSpect[(int)((double)_chromaSpect.Count / (double)width * (double)x)])[(int)(((double)(chroma_size) / (double)(height)) * (double)y)];
```

Na konci vráti vygenerovaný canvas.

4.2.9 Trieda VisualizationsUpdater a jej metódy

Trieda VisualizationsUpdater slúži na vykresľovanie časovej stopy vo vizualizáciách pri prehrávaní. Funguje na jendoduchom princípe, že podľa aktuálneho času zistí miesto vo vizualizácií, kde sa má stopa vykresliť. Uloží si pixely na tomto mieste a toto miesto a prekreslí ich stopou. Pri opakovanom vykreslení najprv vráti pôcodné pixely na miesto pôvodnej stopy a potom vykreslí novú stopu rovnakým spôsobom.

4.2.10 Trieda Settings a FormSettings

Trieda Settings slúži na uchovanie parametrov analýzy. Je serializovateľná, aby sa nastavenia dali exportovať a importovať do súboru. Pri štarte aplikácie sa vytvorí jej inštancia _presets, zavolaním konštruktora bez parametrov, v ktorom sa nastavia predvolené hodnoty.

Trieda FormSettings je trieda formulára v ktorom sa nastavujú parametre a robí export a import parametrov.

4.3 Analýza hudobnej harmónie

V tejto podkapitole rozoberieme implementáciu tried realizujúcich samotnú analýzu. Po zavolaní metódy buttonAnalyse sa vytvorí inštancia triedy *MainAnalyser* a zavolá jej metóda PerformAnalysis, ktorá urobí samotnú analýzu.

4.3.1 Trieda MainAnalyser

Je hlavná trieda na analýzu hudobnej harmónie. Jej konštuktor ako vstup dostane *MemoryStream*, čo je trieda predstavujúca dátový prúd uložený v pamäti, ktorý obsahuje zvukové dáta. Predpokladom je, že dáta prezentujú jeden kanál kódovaný metódou PCM v 16 bitovom vzorkovaní. Ďalšie parametre konštruktora sú vzorkovacia frekvencia, minimálna a maximálna frekvencia, rozlíšenie na koľko dielov chceme vzorkovať jednu oktávu pre CQ transformáciu, faktor na smoothing PCP profilov, minimálna dĺžka segmentu, prekryv FFT okna a parametre heuristiky, koľko najpočetnejších akordov chceme preferovať a faktor ich preferencie.

Hlavné vnútorné prvky tejto triedy sú pomocné triedy, ktoré vykonávajú jednotlivé zložky celého algoritmu:

1. Trieda *FFT* na vykonanie Fourierovej transformácie

2. Trieda *ConstantQT* na vykonanie Konštantnej Q transformácie
3. Trieda *Chromagram* na vytvorenie chromagramu
4. Trieda *ChromaProcess* na manipuláciu s chromagramom a vytvorenie PCP profilu
5. Trieda *PCPMatcher* na priradenie akordu k PCP profilu
6. Trieda *ChordSegmenter* na vytvorenie väčších segmentov z jednotlivých analyzačných úsekov

4.3.2 Hlavný algoritmus – metóda PerformAnalysis

Táto metóda si zaslúži vlastnú podkapitolu, lebo je v nech schovaný celkový návrh algoritmu, ktorý si popíšeme.

Nasledujúce kroky prebiehajú v smyčke, ktorá zabezpečuje postupné načítavanie dát zo streamu ktorý obsahuje zvukové dáta. Dáta sú načítavané v jednotlivých úsekoch (z ang. Frame), tak, aby prebiehal FFT overlapping, teda nejedná sa o čisto sekvenčné čítanie, ale vždy sa istá časť dát načítava znova.

V danej smyčke realizovanej while cyklom prebiehajú samotné kroky algoritmu:

1. Načítajú sa zvukové dáta z dátového prúdu, tieto sú načítané v bytoch, ktoré predstavujú zvukové dáta v 16-bitovom PCM kódovaní. Tieto musíme previesť na jeden zvukový vzorok, takže z dva byty(16 bitov) uloženéz a sebou prevedieme na jeden float. Technicky je to realizované nasledovnou operáciou buffer2[2 * n] & 0xFF) | (buffer2[2 * n + 1] << 8)) / 32768f , kde buffer2 je pole v ktorom je načítaní úsek zo vstupného prúdu. Tento vzorok sa ešte prenásobí hodnotou ktorú vracia Hammingova oknová funkcia – metóda HammingWindow triedy MathUtils, ktorá je potrebná na spresnenie výsledkov Fouriérovej transformácie. Na záver sa vytvorí pole, kde sú tieto vzorky kopírované striedavo s nulami, kde nuly predstavujú imaginárnu časť, lebo vstupom FT sú komplexné čísla.
2. Po spracovaní úseku načítaných zvukových dát sa na tento úsek aplikuje Fourierova analýza, aby sme dostali frekvenčné spektrum zavolaním metódy CalculateFFT triedy FFT.
3. Výsledok Fourierovej analýzy je vstupom pre metódu CalculateCQT triedy ConstantQT na výpočet konštantnej Q analýzy, ktorej výsledkom je spektrum nie lineárne odstupňované, ale logaritmicky, čo je výhodné, lebo tak vieme jednoducho zistiť, ktoré diely zodpovedajú, ktorým tónom. Hudobné tóny sú tiež logaritmicky odstupňované.
4. Z výsledného spektra Konštantnej Q analýzy sa spraví projekcia do jednej oktávy zavolaním metódy MakeChromagram triedy Chromagram, čím dostaneme úsek z chromagramu.
5. Z daného úseku chromagramu sa vyberú špičky zavolaním metódy ProcessChromagram triedy ChromaProcess, tieto sa interpolujú s okolitými hodnotami zavolaním metódy QuadraticInterpolation triedy MathUtils a týmto vytvorí sa konečná verzia 12 poltónového profilu.
6. Na tento poltónový profil sa zavolá metóda Smooth triedy ChromaProcess, ktorá zabezpečí vyhľadzovanie chomagramu. Zároveň sa tento PCP profil uloží do vnútorného prvku _chromaSmoothed Spectrum triedy MainAnalyser.

7. Na daný poltónový profil sa aplikuje detekčný algoritmus, ktorý mu priradí najpravdepodobnejšie akordy, postupným zavolaním metód CalculateCorrelation a GetChord triedy PCPMatcher. Najlepší akord sa uloží do zoznamu akordov chords_d.
8. Vypočíta sa spektrogram pričom dochádza k prevodu dát z Fourérovej transformácie, ktoré sú uložené v komplexnom tvare na amplitúdu v decibeloch. To sa dosiahne nasledovným prepočtom
$$Amplitude(X(c)) = 10 * \log_{10} \sqrt{c.Re^2 + c.Im^2}$$
9. Zavolá akcia Action<int> ProgressCallback, ktorá predá stav analyzovania Progress baru v hlavnom okne.

Počas týchto krokov sa jednotlivé dáta ukladajú aj aby sa v hlavnom okne mohli vizualizovať, konkrétne spektrogram, chromagram pôvodný, chromagram 12 poltónový vyhľadený a špičky z pôvodneho chromagramu.

Nastáva druhá fáza algoritmu. Najprv sa z uložených akordov vyberie podľa parametru _most_used_chords, toľko najpočetnejších akordov detekovaných v prvej fáze algoritmu. Potom prebieha druhá smyčka, ktorá prechádza všetky uložené poltónové profily v predchádzajúcej fáze a robí nasledovné dva kroky:

1. K danému poltónovému profilu deteguje najpravdepodobnejší akord výpočtom korelácie pomocou metódy `CalculateCorrelation` triedy PCPMatcher a následným zavolaním metódy `GetChord`, ktorá podľa korelácie priradí dva najpravdepodobnejšie akordy. Podstatná zmena je, že `CalculateCorrelation` má ako ďalšie parametre vybraté najpočetnejšie akordy a parameter `_used_chords_factor`, ktorým sa znásobuje korelácia u práve najpočetnejších akordov.

2. Daná dvojica najpravdepodobnejších akordov sa zaradí do segmentácie spracovávaného zvukového súboru zavolaním metódy `DoSegmenting` triedy `ChordSegmenter`, ktorá sa ju pokúsi spracovať. Táto metóda ako parameter tiež dostáva zoznam najpočetnejších akordov a podľa neho vyberá, ktorý akord z dvojice vybrať.

Posledná časť tejto metódy už mimo hlavnej smyčky, keď je celý zvukový súbor zanalyzovaný zavolá metódu DoPostProcessing triedy ChordSegmenter na finálnu úpravu segmentácie akordov, vymazaním veľmi krátkych segmentov a zľúčením rovnakých susediacich segmentov.

4.3.3 Trieda FFT

Jedná sa o triedu na výpočet diskrétnej Fouriérovej analýzy. Realizovaná je pomocou algoritmu FFT – Fast fourier analysis. Samotná analýza je realizovaná jednou statickou funkciou CalculateFFT, ktorá má na vstupe pole typu double, alebo float s dátami, veľkosť transformovaného úseku a znamienko určujúce, či chceme robiť FT alebo inverznú FT. Vstupné pole musí mať dvojnásobnú veľkosť transformovaného úseku. Hodnoty vo vstupnom poli predstavujú komlexné čísla, takže dva prvky pola predstavujú jedno číslo. Prvý prvok je reálna časť a druhý imaginárna.

Detailnú implementáciu nebudem popisovať, bližšie informácie k použitej implementácií nájdete v knihe Numerical Recipes in C: The Art of Scientific Computing [9].

4.3.4 Trieda ConstantQT

Jedná sa o triedu na výpočet Konštantnej Q transformácie. Konštruktor tejto triedy dostáva ako parametre vzorkovaciu frekvenciu, minimálnu a maximálnu frekvenciu z ktorej bude počítať transformáciu a počet dielov v jednej oktáve. Podľa týchto dát si v konštruktore spočíta potrebnú veľkosť okna Fouriérovej transformácie, ktorá sa nastaví prvku _frame_size triedy MainAnalyzer v konštruktore.

Cieľom tejto triedy je vytvoriť logaritmicky odstupňované spektrum, ktoré sa dá jednoducho namapovať na poltóny. Implementácia tejto triedy je založená na algoritme efektívneho výpočtu tejto transformácie uverejnenom v [4], kde nájdete detailnejšie informácie.

4.3.5 Trieda Chromagram

Trieda Chromagram vykonáva projekciu výsledného spektra konštantnej Q transformácie do jednej oktávy. Obsahom je jedna statická metóda MakeChromagram, ktorá dostane na vstup pole s výsledkami triedy ConstantQ, a ako parameter počet dielov oktávy. Jedným priechodom vstupného pola vytvorí výslednú oktávu tak, že jednak prevedie komplexné čísla, ktoré sú vo vstupnom poli na amplitúdu a že sčítava jednotlivé diely, ktoré predstavujú rovnaký diel, len v inej oktáve.

4.3.6 Trieda ChromaProcess

Trieda ChromaProcess vykonáva operácie s výstupom triedy Chromagram, ktorý predstavuje pole reprezentujúce oktávu rozdelenú na diely, čo sú vlastne prvky tohoto poľa. Okrem toho má funkcie na vytváranie PCP profilov a manipuláciu s nimi.

Základná funkcia je ProcessChromagram, ktorá v poli reprezentujúcom jednu oktávu rozdelenom na niekoľko dielov nájde lokálne maximá. Tieto lokálne maximá by mali predstavovať znejúce poltóny. Na každé lokálne maximum sa aplikuje kvadratická interpolácia zavolaním metódy *QuadraticInterpolation* z triedy *MathUtils*. Interpolujú sa maximum a jeho dva susedné prvky. Z výsledného polynómu sa zoberie maximum, ktoré by malo reprezentovať znejúci poltón. Pred vrátením týchto lokálnych maxím sa ešte premapuje ich pozícia do 12 poltónového chromagramu, lebo pôvodné pole môže byť väčšie, typicky 36 dielov na oktávu.

Funkcia SimpleProcess vykonáva rovnakú funkciu ako ProcessChromagram, ale nerobí interpoláciu.

Funkcia MakePCP slúži na vytvorenie PCP profilu. Na vstupe dostane lokálne maximá poltónov, čo sú výstupy funkcií ProcessChromagram a SimpleProcess, ako zoznam dvojíc (pozícia poltónu, amplitúda) a vytvorí z nich pole 12 poltónov a ich amplitúd, ktoré predstavuje PCP profil.

Funkcia Smooth slúži na vyhľadzovanie PCP profilov v čase. Kedže samotný algoritmus spracováva zvukové dáta po častiach, tak v niektorých úsekoch môže vyhľadzovanie PCP profilov dávať lepšie výsledky ako bez neho. Sú to hlavne úseky počas znenia jednej harmónie, kde vyhľadenie robí stabilnejšie výsledky. Naopak trocha stiera hranice pri zmene harmónie, kde môže dôjsť k chybným predpovediam.

Samotné vyhľadzovanie sa nastavuje konštantou *_smoothing_factor,* ktorá znamená, koľko percent z predchádzajúceho profilu sa prenesie do nasledujúceho.

V triede ChromaProcess je vnútorný prvok reprezentujúci PCP profil, cez ktorý sa tieto hodnoty prenášajú. Výsledkom tejto funkcie je vrátenie tohoto prvku.

4.3.7 Trieda PCPMatcher

Trieda PCPMatcher slúži na priradenie daného akordu, resp. harmonickej značky k vstupnému PCP profilu.

Jej prvkami je zoznam typov akordov, kde sú uložené vzorové PCP profily, názov akordu a počet možných základných tónov.

```
List<ChordType> _chord_types = new List<ChordType> {
new ChordType(new double[]{1,1,1,1,1,1,1,1,1,1,1,1}, "None",        1),
new ChordType(new double[]{1,0,0,0,1,0,0,1,0,0,0,0}, "Major",      12),
new ChordType(new double[]{1,0,0,1,0,0,0,1,0,0,0,0}, "Minor",      12),
new ChordType(new double[]{1,0,0,0,1,0,0,1,0,0,0,1}, "Major7",     12),
new ChordType(new double[]{1,0,0,1,0,0,0,1,0,0,0,1}, "MinorMajor7",12),
new ChordType(new double[]{1,0,0,1,0,0,0,1,0,0,1,0}, "Minor7",     12),
new ChordType(new double[]{1,0,0,0,1,0,0,0,1,0,0,0}, "Augmented",   4),
new ChordType(new double[]{1,0,0,1,0,0,1,0,0,0,0,0}, "Diminished", 12),
new ChordType(new double[]{1,0,0,1,0,0,1,0,0,0,1,0}, "Diminished7",12),
new ChordType(new double[]{0,0,0,0,0,0,0,0,0,0,0,0}, "0", 0) }
```

Typov akordov je viac, ale po odskúšaní v praxi som zistil, že rozpoznávanie nebolo celkom spoľahlivé a znížením počtu typov akordov sa zlepšila presnosť, rozpoznávania. To je spôsobené aj tým, že akordy, ktoré nerozpoznávame, sa vyskytujú zriedkavo.

V konštruktore sa vygenerujú všetky typy akordov, cyklickým posununtím vzorového PCP profilu, podľa základného tónu, čím sa vytvrí vzorový PCP profil konkrétneho akordu. Akordy sú uložené ako inštancie triedy `Chord`, ktorá ešte obsahuje, základný tón, vyjadrenie typu pomocou inštancie triedy ChordType a počet tónov tohto akordu.

Metóda CalculateCorrelations ako parameter dostane PCP profil a vráti pole korelácií ku všetkým akordom. Počíta sa ako vážený skalárny súčin dvoch vektorov, v našom prípade PCP profilov. $Score(p) = \sum_{p=0}^{11} W_c(p) * PCP(p)$

Zároveň má táto metóda druhú definíciu, kde má naviac parameter zoznam najpočetnejčích akordov uložený v type HashSet a parameter zosilnenia korelácie pri týchto najpočetnejších akordoch most_chords_factor. Týmto parametrom

prenásobí vypočítanú koreláciu profilu s profilom akordu, ak je medzi najpočetnejšími.

Metóda GetChord berie sko vstup zoznam korelácií vytvorených predchádzajúcou metódou a vráti dvojicu dvoch najväčších ako indexy, ktoré sú rovnaké s indexami v zozname akordov, takže sa jedná o indexy dvoch najpravdepodobnejších akordov.

4.3.8 Trieda ChordSegmenter

Trieda ChordSegmenter slúži na dlhších vytvorenie segmentov harmónie zo zoznamu vypočítaných poltónových profilov. Väčšina jej metód je čisto technických na tvorbu týchto segmentov, takže si popíšeme len hlavnú metódu DoSegmenting.

Tá dostane ako parameter aktuálny index(frame) profilu, ktorý slúži zároveň na označovanie času, lebo celkový počet spracovávaných úsekov(frameov) je dopredu vypočítaný a aj všetky dáta v procese analýzy sú indexované práve číslom úseku. Takže čas vieme jednoducho vypočítať prepočtom celkovej dĺžky skladby na celkový počet úsekov.

Ďalšie parametre sú pole korelácií, dvojica akordov s najväčšou koreláciou a zoznam najpočetnejších akordov.

Tvorba segmentu prebieha tak, že sa ajprv podľa korelácií zvolí konečne rozpoznaný akord. Ak sú hodnoty korelácie pod prahovými hodnotami tak sa žiadny akord nevyberie. Inak sa vyberie akord s najvyššou koreláciou. Ak sa nenachádza medzi najpočetnejšími akordami, ale zároveň sa tam nachádza druhý najlepší akord, tak sa zvolí ten. Heuristika sa dá jednoducho vypnúť nastavením počtu vybraných najpočetnejších akordov na nulu.

Samotný proces tvorby segmentov prebieha stavovo, kde sa kontroluje či je nejaký segment rozrobený, alebo nie. Či sa zhoduje vybraný akord z rozrobeným segmentom a pod. Podľa toho sa segmenty predlžujú, vytvárajú a zatvárajú.

Posledná dôležitá metóda je DoPostProcessing, ktorá podľa parametra minimálnej dĺžky segmentu vymaže krátke segmenty a spojí susediace segmenty s rovnakými akordami.

4.3.9 Trieda Segmentation

Trieda Segmentation slúži na uloženie jednotlivých segmentov harmónie v jednej skladbe. Tie sú uložené v type List<Segmentaion> _segments, obsahuje metódy Insert, Strech, Remove na manipuláciu so segmentami. Tieto metódy sú jednoducho realizované, takže ich bližšie nebudem popisovať. Ešte obsahuje metódy Neighbours, SameAndNeighbours, FrameInSegment a FrameInSegmentation. Prvé dve porovnávajú dva segmenty, či susedia a sú rovnaké. Druhé dve zisťujú, či sa daný úsek nachádza v segmente resp. celej segmentácií. Ak sa nachádza pomocou property IdentifiedSegmentIndex resp. IdentifiedSegmentChord zistíme index tohoto segmentu resp. jeho harmóniu. Poslednou metódou je OnSerializing s atribútom [OnDeserializing], ktorá sa volá pri deserializácií a zavolá konštruktor na _segments, lebo serializátor to nerobí automaticky.

4.3.10 Pomocné triedy Chord, ChordType, Segment, TimeSegment

Spoločným znakom týchto tried je, že slúžia primárne na ukladanie dát a neobsahujú žiadne zložité metódy.

Trieda ChordType slúži na reprezentáciu typu akordu. Pod typom myslíme, aké sú hodbné intervaly tónov, z ktorých sa daný akord skladá.

Trieda Chord slúži na reprezentáciu akordu.

Triedy Segment a TimeSegment na reprezentáciu úseku skladby s jednou harmóniu. Ich vnútorné prvky sú začiatok segmentu a koniec segmentu a daný akord harmónie.

4.3.11 Trieda MathUtils

Trieda MathUtils je pomocná trieda, v ktorej sú implementované matematické funkcie, ktoré nie sú dostupné v štandardnej knižnici.

Funkcia NextPow2 vracia najbližšiu druhú mocninu k vstupnému číslu.

Funkcia HammingWindows je Hammingova oknová funkcia, slúžiaca pri Fouriérovej transformácií.

Funkcia HannWindow je Hannova oknová funkcia, ktorá slúži ako alternatíva k Hammingovej funkcií.

Funkcia QuadraticInterpolation robí kvadratickú interpoláciu bodov [0,y0],[1,y1], [2,y2] a vracia lokálne maximum/minimum interpolovaného polynómu.

4.4 Prehrávanie

Prehrávanie zvukových dát je realizované pomocou knižnice NAudio. K samotnému prehrávaniu je vytvorených niekoľko pomocných tried, ktoré nám umožňujú kontrolu samotného procesu prehrávania.

4.4.1 Interface IAudioPlayer a trieda AudioPlayer

Interface IAudioPlayer je základný interface, z ktorého sa odvádzajú triedy na prehrávanie zvukových súborov. Tento interface má nasledujúce metódy a vlastnosti.

Metódy LoadFile, Play, Stop slúžia na načítanie súboru a prehratie a zastavenie prehrávania. Členy interfacu sú CurrentPosition, EndPostition, StartPosition, ktoré reprezentujú časové značky.

Trieda AudioPlayer je hlavná trieda na prehrávanie zvukových súborov, ktorá implementuje IAudioPlayer.

Jej hlavné členy okrem členov deklarovaných v interface-y sú inštancie tried WaveOut waveOut, TrimWaveStream inStream, SampleAggregator sampleAggregator a WaveFormat fileFormat.

Trieda WaveOut zabezpečuje abstrakciu nad zvukovou kartou. TrimWaveStream robí abstrakciu nad zvukovými dátami. WaveFormat je trieda na uloženie parametrov zvukových dát v triede WaveStream, ako napríklad počet kanálov, vzorkovacia frekvencia a počet bitov v jednom vzorku.

Okrem týchto členov má trieda AudioPlayer inštanciu triedy PlaybackState a prislúchajúce metódy, ktoré slúžia manažovanie stavov, či sa zrovna nejaký súbor prehráva, aby nebolo možné spustiť prehrávanie počas prehrávania.

Metóda LoadFile, zresetuje WaveOut a TrimWaveStream zavolaním metód CloseWaveOut a CloseInStream. Potom sa inicializuje TrimWaveStream podľa toho či je vstup súbor vo formáte WAV alebo MP3, kde pri MP3 sa použije metóda ConvertPcmStream obsiahnutá v knižnici NAudio, ktorá nám skonvretuje tento súbor na WaveStream. Potom sa vytvorí inštancia WaveOut pomocou metódy CreateWaveOut, ktorá nastaví WaveOutu dátový prúd inStream.

Metóda Play nastaví začiatočnú pozícu prúdu, nastaví stav na prehrávanie a zavolá metódu waveOut.Play, ktorá začne čítať z dátového prúdu a tieto dáta posielať do zvukovej karty.

4.4.2 Trieda TrimWaveStream

Trieda TrimWaveStream slúži na abstrakciu zvukových dát vo forme dátového prúdu (preklad z ang. Stream). Dátové prúdy sú koncept, ktorý umožňuje efektívne čítanie a zapisovanie dát do a z nejakého zariadenia(napr. súbor, alebo pamäť).

Táto trieda je odvodená od triedy WaveStream z knižnice NAudio. Rozdiel medzi nimi je ten, že umožňuje prehrávať len nejaký výsek z daného zvukového súboru, lebo obsahuje premenné kde sa dá uložiť požadovaný interval. Tiež obsahuje inštanciu triedy SampleAggregator, ktorá slúži na vyvolanie udalosti pri prehrávaní v pravidelných intervaloch.

Základnou metódou tejto triedy je metóda Read, ktorá slúži na čítanie dát zo zdrojového súboru. Jedná sa o štandardný princíp čítania z dátového prúdu, kde ako parameter dostane táto metóda buffer, do ktorého sa majú dáta načitať, pozícu a počet bytov, koľko chceme načítať. Vráti nám počet načítaných bytov. Špecifické v našej metóde Read je, že zároveň z načítaných bytov zvukového súboru vypočíta jednotlivé vzorky, predpokladá sa 16-bitové kódovanie PCM vstupného zvukového súboru, tak že z dvoch bytov, ktoré majú dokopy 16 bitov, vyrobí jeden vzorok uložený do float-u a pošle ho do inštancie SampleAggregator.

4.4.3 Trieda SampleAggregator

Trieda SampleAggregator slúži na akumuláciu vzorkov a vyvolanie udalosti po presiahnutí určitého počtu vzorkov. Zároveň z týchto vzorkov môže počítať nejaké dáta, v našom prípade maximálny a minimálny vzorok. Toto sa využíva pri vykreslovaní priebehu prehrávania zvukového súboru, kde je nežiaduce, aby sa vykresloval každý jeden vzorok. Technicky to funguje tak, že obsahuje metódu Add na pridanie vzorku, ktorá zároveň počíta maximum a minimum a počet pridaných prvkov. Pri presiahnutí počtu vzorkov sa vyvolá udalosť cez delegát MaximumCalculated typu EventHandler<MaxSampleEventArgs>. MaxSampleEventArgs je trieda odvodená od EventArgs, ktorá slúži na predávanie parametrov cez udalosti. V našom prípade sa predáva maximálny a minimálny vzorok. Po vyvolaní tejto udalosti sa vnútorné hodnoty zresetujú a znova sa počíta maximum a minimum z prichádzajúcich vzorkov.

4.5 Export a import výsledkov analýzy

Exportovanie a importovanie výsledkov analýzy je realizované pomocou serializácie a do formátu JSON. Serializácia je vykonávaná triedou zo štandardnej knižnice MS .NETu System.Runtime.Serialization.Json.DataContractJsonSerializer. Táto trieda serializuje triedy označené atribútom [DataContract] a jednotlivé jej prvky atribútom [DataMember].

4.5.1 Trieda AnalysationDataSpecific

Trieda AnalysationDataSpecific slúži na serializáciu dát, tak že údaje v nej sú použiteľné len v tejto aplikácií, lebo jednotlivé úseky harmónie sú zaznamenané v inštancií triedy Segmentation. To znamená, že časy v jednotlivých segmentoch sú uložené ako indexy daných úsekov(frame) pri analyzovaní. Akordy sú zase uložené ako indexy v zozname akordov triedy PCPMatcher.

4.5.2 Trieda AnalysationDataGeneral

Trieda AnalysationDataGeneral slúži na serializáciu analyzačných dát v prirodzenom ponímaní. Technicky sú jednotlivé úseky harmónie uložené ako inštancia List<TimeSegmentation>. To znamená, že jednotlivé segmenty harmónie sú označené čitateľným časom a akordom.

4.6 Porovnávanie výsledkov analýzy

Porovnávanie výsledkov anaýzy je realizované v jednej triede. Jedným jednoduchým algoritmom. Implementoval som ju, lebo som ju mal v zadaní.

4.6.1 Trieda AnalysationDataComparer

Trieda AnalysationDataComparer slúži na porovnávanie dvoch výsledkov analýz.

Predpokladané využitie je, že jeden výsledok je výstup analýzy programu a druhý je naimportovaný užívateľom, ktorý si ho ručne upravil. Toto je ale len jeden účel na ktorý môže byť táto trieda použitá, druhý môže byť na zistenie istej harmonickej podobnosti dvoch rozdielnych skladieb.

Konštruktor tejto triedy má ako vstupný parameter typ *AnalysationDataSpecific*, ktorý predstavuje prvý súbor dát, ktoré sa budú porovnávať. Tento parameter si potom uloží ako vnútorný člen *_data* tejto triedy. Ide o dáta vytvorené analýzou počas behu programu.

Hlavná metóda tejto triedy je *CompareWithGeneral*, ktorá má ako vstupný parameter typ *AnalysationDataGeneral*, čo predstavuje druhý súbor dát na porovnávanie. Tieto dáta získame importovaním z externého súboru. Potom prebehne algoritmus samotného porovnávania, ktorý vypočítava dve hodnoty. Funguje tak, že najskôr skonvertuje vstupný parameter na *AnalysationDataSpecific*, ktorý uloží ako vnútornú premennú *converted_data.* Pri konverzií sa zabezpečí, aby tento

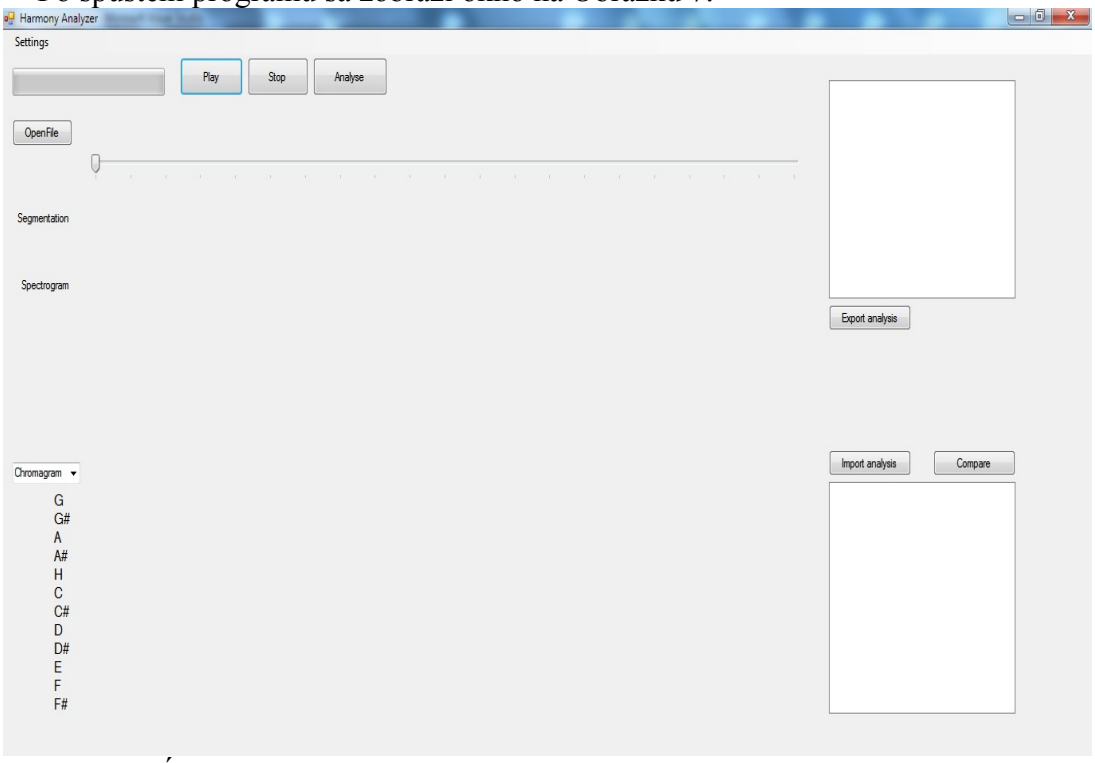
konvertovaný typ mal rovnakú veľkosť(člen Total_frames). Samotné porovnávanie prejde všetky úseky(z ang. frames) a v každom súbore dát zisťuje, či sa daný úsek nachádza v nejakom rozpoznanom harmonickom segmente. Ak áno, tak sa uloží akord tohto segmentu. Na záver sa porovnajú tieto akordy. Ak sú rovanké, tak sa zväčšia obidve hodnoty *simple_overlap* aj *overlap_score* o jedna. Hodnota *overlap_score* sa zvyšuje aj v prípade rozdielnych akordov, ale rovnakého základného tónu, ale len o 0,5. Funkcia vráti dvojicu pomerov týchto hodnôt k celkovému počtu úsekov v percentách.

5 Užívateľská dokumentácia

V tejto kapitole rozoberieme užívateľskú dokumentáciu, ako návod na obsluhu aplikácie pre užívateľa.

5.1 Základná funkcionalita

Po spustení programu sa zobrazí okno na Obrázku 7.



Obrázok 7: Úvodná obrazovka

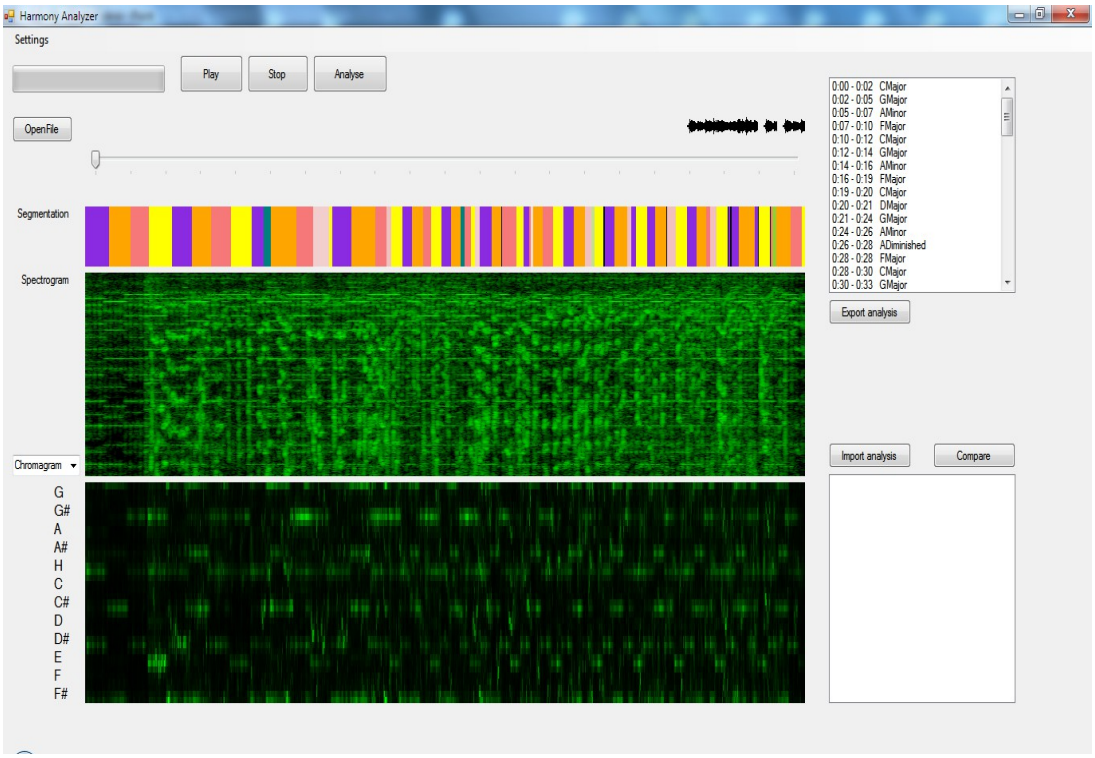
Na začiatku má užívateľ 3 možnosti: Načítať zvukový súbor stlačením tlačidla OpenFile, Vyvolať okno nastavení stlačení menu ponuky Settings a Importovať analýzu stlačením tlačidla ImportAnalysis.

5.1.1 Načítanie súboru a následné možnosti

Po stlačení OpenFile sa zobrazí štandardný dialóg systému MS Windows, v ktorom si vyberie zvukový súbor. Na výber má možnosť formáty WAV a MP3.

Potom môže obsluhovať prehrávanie súboru tlačidlami Play a Stop.

Tlačidlo Analyse slúži na spustenie samotnej analýzy. Počas nej sa zablokujú niektoré tlačidlá a zastaví prípadné bežiace prehrávanie. Indikácia analýzy sa zobrazuje v ukazovateli v ľavom hornom rohu. Po skončení analýzy sa vykreslia niektoré dáta. Obrazovku programu vidíme na Obrázku 8.



Obrázok 8: Obrazovka po analýze

Prvá vizualizácia s popisom Segmentation ukazuje rozdelenie úsekov hudobnej harmónie v danej skladbe. Farby sú generované deterministicky, ale určené sú skôr len na zbežnú orientáciu, preto nie je nikde zobrazené priradenie farieb k akordom.

Druhá vizualizácia s popisom Spektrogram zobrazuje frekvenčné spektrum v rozsahu 0-2100Hz pri načítaní 44100Hz vzorkovaného súboru, čo je predpoklad. Ide len o orientačnú vizualizáciu.

Tretia vizualizácia pri zvolení popisu Chromagram zobrazuje frekvenčné spektrum namapované do jednej oktávy, ktorá je rozdelená podľa nastavenia parametru Octave bins, čo je počet dielikov.

Po prepnutí popisu na Smoothed sa zobraza podobné dáta, ako v popise Chromagram, ale oktáva je rozdelená na 12 dielikov, ktorých popis určuje stupnica vľavo. Užívateľ z neho môže vidieť intenzitu jednotlivých poltónov na danom mieste.

V pravom hornom rohu môže užívateľ vidieť jednotlivé segmenty harmónie v zozname.

Ak stlačí tlačidlo Play súbor sa začne prehrávať v prvý dvoch obrázkoch bude práve prehrávané miesto indikovať sledovacia čiara. Zároveň sa bude v hornej časti okna vedľa tlačidiel zobrazovať časová stopa prehrávania a aktuálna hudobná harmónia.

Pri prehrávaní sa ešte zobrazuje vlnový priebeh skladby. A aktuálna časová stopa na posuvníku. Pri posunutí posuvníka sa zmení prehrávaná pozícia súboru.

Tlačidlo Export analysis exportuje analýzu do súboru vo formáte JSON, ktorý je síce čitateľný, ale pri jeho úpravách sa musí dodržiavať formát. Tlačidlo Import analysis importuje analýzu zo súboru a zobrazí ju v zozname pod ním. Tlačidlo Compare porovná analýzy a zobrazí dve hodnoty. Prvá určuje % prekryv rovnakej

harmónie. Druhé číslo započítava aj akordy s rovnakým základným tónom ale rozdielneho typu.

5.1.2 Nastavenie parametrov

Po stlačení Settings sa zobrazí okno na nastavenie parametrov.

Dané parametre sa dajú načítať a uložiť do súboru tlačidlami Load resp. Save settings. Tlačidlom OK sa potvrdia zmenené hodnoty. Pri zadaní zlých hodnôt sa zobrazí chyba.

Popis jednotlivých parametrov je takýto.

Correlation threshold určuje nakoľko veľká musí byť korelácia rozpoznávaného akordu , aby sa zaradil do analýzy.

Octave bins určuje počet dielov v jednej oktáve chromagramu.

Minimal a maximal frequency sú hranice frekvencie z ktorej sa bude počítať chromagram.

FFT overlap určuje prekryv Fourierovej analýzy, väčšia hodnota znamená väčší prekryv.

Smoothing factor nastavuje rozmazávanie chromagramu.

Minimal segment length je minimálna dĺžka harmonického segmentu.

Most used chords amplification nastavuje preferovanie najpočetnejších akordov. Odporúčam nastaviť na cca 1,4.

Most used chords count nastavuje koľko najpočetnejších akordov sa má preferovať.

6 Záver

6.1.1 Kvalita programu a konkurenčné programy

Program robí danú úlohu subjetívne dobre. Lepšie výsledky dosahuje na inštrumentálnych skladbách. Najlepšie na hudobných doprovodoch Pop music. Horšie výsledky dosahuje na bežných skladbách Pop music, ale zmenami parametrov sa dajú výsledky vylepšiť. Najmä keď daná skladba nepoužíva veľa akordov a tieto za začnú preferovať. Nastavenie 4-6 preferovaných akordov a zosilnenie cca 1,4.

Za konkurenčné programy považujem jednak programy, ktoré boli vytvorené vo výskumných článkoch, ale tieto sú neverejné.

Potom je to program SonicVisualizer [http://www.sonicvisualiser.org/], ktorý danú úlohu, ale nerieši. Až plugin do tohoto programu Chordino and NNLS Chroma [http://www.vamp-plugins.org/download.html]. Možno rieši moju úlohu aj niektorý z ďalších pluginov, ale to som nezisťoval.

Myslím si, že tento plugin je zložitejší a dáva lepšie výsledky, lebo implementoval Hidden markov model do detekčného algoritmu.

Implementovaná aplikácia dáva uspokojivú mieru výsledkov a výhoda je, že ide o integrovaný celok. Nainštalovanie tohto pluginu nie je intuitívne a spočíva vo vytvorení adresára v systémovom adresári Program Files a nakopírovaní súborov do neho.

Druhá výhoda implementovanej aplikácie je, že umožňuje export dát analýzy. To som si v tomto plugine ani programe nevšimol.

Ďalší program o ktorom som sa dozvedel až na záver písania tejto práce je HPA [https://patterns.enm.bris.ac.uk/hpa-software-package], ktorý má podobnú funkcionalitu ako Chordino a je k dispozícií aj so samostatným GUI, ktoré sa podobá mojej aplikácií. Obidva konkurenčné programy sú písané v jazyku C++.

Jedna z výhod implementovanej aplikácie je objektová štruktúra, ktorá umožňuje rozširovať funkcionalitu algoritmu a dobre okomentovaný kód.

6.1.2 Záver

Jednotlivé vety zadania práce zneli:

1. Cieľom je navrnúť a implementovať systém na analýzu zvukovej harmónie.
2. Súčasťou práce je teoretický popis metód a zdôvodnenie ich použitia.
3. Výstupom tejto analýzy bude automatické rozpoznanie akordov v hudobnej skladbe.
4. Formát vstupu je súbor WAV a formát výstupu sú časové intervaly skladby s harmonickou značkou.
5. Výstup bude vizualizovaný v GUI, ale tiež ho bude možné exportovať do textového súboru
6. Súčasťou práce bude modul na porovnanie úspešnosti detekcie.

Všetky body som splnil, rovnako aj veci napísané v úvodnej podkapitole 1.3 Cieľ práce.

7 Zoznam použitej literatúry

[1] http://dai.fmph.uniba.sk/~filit/fvt/teoria_hudobna.html (online, 21. 7. 2014)

[2] Fujishima, T. (1999). Realtime chord recognition of musical sound: A system using common lisp music. In Proc. ICMC, pages 464–467, Beijing.

[3] C. A. Harte and M. B. Sandler. Automatic chord identification using a quantised chromagram. In *Proceedings of the 118th Convention of the Audio Engineering Society*,Barcelona, Spain, May 28-31 2005.

[4] Benjamin Blankertz, The Constant Q Transform http://wwwmath.uni-muenster.de/logik/Personen/blankertz/constQ/constQ.html (online, 21.7.2014)

[5] Wikipedia A/D převodník http://cs.wikipedia.org/wiki/A/D_p%C5%99evodn%C3%ADk (online 29.7.2014)

[6] Chafe, Chris, et al. *Techniques for note identification in polyphonic music*. CCRMA, Department of Music, Stanford University, 1985.

[7] Chafe, Chris, and David Jaffe. "Source separation and note identification in polyphonic music." *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'86.*. Vol. 11. IEEE, 1986.

[8] Yushi Aono, Haruhiro Katayose, and Seiji Inokuchi, A Real-time Session Composer with Acoustic Polyphonic Instruments, Proceedings of ICMC 1998, pp. 236-239, 1998.

[9] Numerical Recipes in C: The Art of Scientific Computing chapter 12.2 http://cacs.usc.edu/education/phys516/c12-2.pdf (online 31.7.2014)

[10] Skripta k Algoritmům a datovým strukturám: Martin Mareš a Tomáš Valla http://mj.ucw.cz/vyuka/ads/44-fft.pdf (online 31.7.2014)

[11] http://en.wikipedia.org/wiki/Constant_Q_transform (online 31.7.2014)

[12] Spectrum and spectral density estimation by the Discrete Fouriertransform (DFT), including a comprehensive list of windowfunctions and some newat-top windows.: G.Heinzel, A.Rudigerand R.Schilling , Max-Planck-Institut fur Gravitationsphysik(Albert-Einstein-Institut) Teilinstitut Hannover http://holometer.fnal.gov/GH_FFT.pdf (online 31.7.2014)

[13] J.P. Bello and J. Pickens, A Robust Mid-Level Representation for Harmonic Content in Music Signals. ;In Proceedings of ISMIR. 2005, 304-311.

8 Prílohy

8.1.1 Adresárová štruktúra sprievodného CD

-praca: obsahuje túto prácu

-testovacie_subory: obsahuje niektoré súbory použité při ľadení

-HarmonyAnalyser: obsahuje zdrojové kódy, spustiteľný kód a generovanú programátorskú dokumentáciu