Harmonija u realnom vremenu

Srnec, Filip

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO–MATEMATIČKI FAKULTET
MATEMATIČKI ODSJEK

Filip Srnec

HARMONIJA U REALNOM VREMENU

Diplomski rad

Voditelj rada:
izv. prof. dr. sc. Saša Singer

Zagreb, rujan 2017.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Za dedu

Sadržaj

Uvod

Zamislimo solo pjevača kako izvodi neku skladbu. Neovisno o kvaliteti izvedbe, solo pjevač ne može u istom trenutku pjevati više različitih *melodija*. On ima samo jedan glas, koji u određenom vremenskom intervalu, može proizvoditi samo jedan *ton*. Zamislimo sada da se pjevaču pridruži još nekoliko drugih pjevača te da svi zajedno čine zbor. Svaki od pjevača u zboru može pjevati vlastitu melodiju te na taj način početna skladba dobiva potpuno novi karakter. Na taj način, pjevači mogu stvarati razne *harmonije* koje doprinose punini i efektnosti skladbe koju izvode.

U širem smislu, izraz harmonija u glazbi predstavlja spajanje više tonova zajedno, koji tada čine *akorde*, skupove od više tonova koji se pojavljuju istodobno. Osnovno pitanje kojim se bavimo u ovom radu je možemo li automatski stvarati harmoniju pomoću računala. Konkretnije, proučavamo način na koji iz ulaznog glazbenog signala, koji opisuje jednu melodiju, stvaramo više različitih melodija koje se međusobno upotpunjuju. Na taj način, u slučaju ljudskog glasa, postižemo efekt zborskog pjevanja. U slučaju glazbenog instrumenta, simuliramo istovremeno sviranje više (istovrsnih) instrumenata.

Harmonizacija se, na način kako je opisana u ovom radu, ostvaruje kombiniranjem melodija koje su dobivene pomakom za određeni interval (frekvencijskim pomakom početnog signala) u odnosu na zadanu početnu melodiju. Tehnike koje se koriste za takve frekvencijske pomake ulaznog signala jednim se imenom nazivaju *Pitch Shifting* algoritmi. *Pitch Shifting* algoritam kojeg predstavljamo u ovom radu, bazira se na poznatoj tehnici u digitalnoj obradi signala koja se u literaturi naziva *Phase Vocoder*. Ona se sastoji od nekoliko faza u kojima glavnu ulogu ima efikasna implementacija *diskretne Fourierove transformacije*. Diskretna Fourierova transformacija omogućuje rastav početnog signala na pripadne parcijalne frekvencijske komponente, što omogućuje razne modifikacije u frekvencijskoj domeni.

Glavni cilj ovog rada je izraditi programsko rješenje za automatsku harmonizaciju zvučnog signala u realnom vremenu. To znači da je cilj iz početnog ulaznog signala dobiti harmonizirani signal koji se šalje na audio izlaz na način da, iz korisnikove perspektive, nema nikakvih zvučnih devijacija u smislu prekida signala ili pojave disonantnih tonova. Cjelokupno procesiranje određenog dijela ulaznog signala, okvira

koji se obrađuje u jednoj iteraciji algoritma, mora se obaviti prije nego što završi reprodukcija prethodnog okvira. Iz tog razloga, naglasak stavljamo na efikasnost implementacije algoritma u smislu vremenske složenosti. Bitan dio naših razmatranja je i automatsko generiranje harmonije na način da krajnji rezultat ima smisla u glazbenom kontekstu u kojem ga promatramo.

U poglavlju 1 definirat ćemo neke osnovne pojmove iz područja digitalne obrade signala i teorije glazbe, koji će nam biti potrebni u daljnjim razmatranjima. Nadalje, u poglavlju 2 bavit ćemo se diskretnom Fourierovom transformacijom koja je osnovna matematička podloga za promatrani algoritam, te ćemo predstaviti efikasnu implementaciju algoritma za računanje diskretne Fourierove transformacije. Poglavlje 3 predstavlja centralnu točku ovog rada. U tom poglavlju detaljno je razrađen *Pitch Shifting* algoritam kojeg koristimo u konačnom programskom rješenju. U završnom poglavlju (poglavlje 4) donosimo osnovne funkcionalnosti i detalje vezane uz implementaciju programskog rješenja za harmonizaciju ulaznog signala u realnom vremenu, te komentiramo njegove performanse u smislu vremenske složenosti.

Poglavlje 1

Signali i glazba

U ovom poglavlju opisat ćemo osnovne pojmove vezane uz digitalnu obradu signala i teoriju glazbe, koje ćemo koristiti u daljnjim poglavljima.

1.1 Općenito o signalima

Signal se, općenito, može promatrati kao apstraktni matematički opis nekog promatranog fizikalnog procesa. Obično su to funkcije koje opisuju promatranu fizikalnu varijablu ili fizikalni proces, odnosno, nose neku informaciju o tom sustavu ili procesu. Cilj obrade signala je, iz danog signala, izvući i obraditi tražene informacije. Primjeri signala s kojima se često susrećemo u svakodnevnom životu su ljudski govor, glazba, slika ili video signal. Većina signala s kojima se susrećemo generirani su prirodnim putem, no signali, također, mogu biti generirani sintetički, ili kao rezultat računalne simulacije. Dakle, signal, općenito, možemo promatrati kao funkciju više nezavisnih varijabli.

Ovisno o prirodi nezavisnih varijabli i vrijednostima koje poprima funkcija koja definira signal, moguće je definirati nekoliko vrsta signala. Konkretno, ovisno o domeni pripadajuće funkcije, signal može biti kontinuiran ili diskretan. Nadalje, signal može biti realan ili kompleksan. Također, razlikujemo jednodimenzionalne, dvodimenzionalne i višedimenzionalne signale.

Primjeri jednodimenzionalnog signala su ljudski govor ili zvučni (audio) signal. Naime, zvuk možemo promatrati kao brzu promjenu tlaka zraka u vremenu. Primjer dvodimenzionalnih signala su crno bijele slike, jer imaju svoju horizontalnu i vertikalnu komponentu, vrijednosti pojedinih piksela.

Budući da je tema ovog rada direktna primjena digitalne obrade signalu u glazbenoj domeni, koncentrirat ćemo se na jednodimenzionalne signale te ćemo, u nastavku rada, pod pojmom signala podrazumijevati upravo jednodimenzionalni signal. Sma-

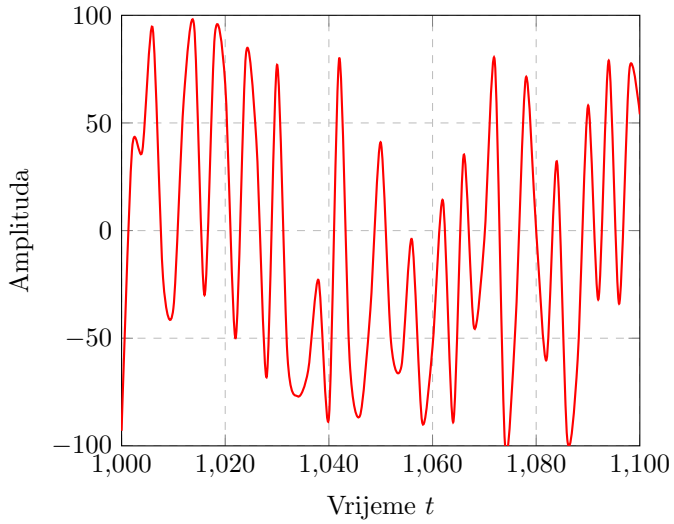
trat ćemo da on ovisi o varijabli koja reprezentira vrijeme, odnosno, neki određen vremenski trenutak. Za detaljniji uvid u pojam signala i osnovne principe digitalne obrade signala upućujemo čitatelja na [1] i [15].

U nastavku, donosimo formalne definicije kontinuiranog, odnosno, diskretnog signala.

Definicija 1.1.1. *Neka su $t_1, t_2 \in \mathbb{R}$ takvi da je $t_1 < t_2$.* ***Kontinuirani signal*** *je svaka funkcija $x : [t_1, t_2] \to \mathbb{R}$.*

Definicija 1.1.2. *Neka su $n_1, n_2 \in \mathbb{Z}$ takvi da je $n_1 < n_2$ i neka je $T \in \mathbb{R}$ takav da je $T > 0$. Neka je $K = \{nT \mid n \in \mathbb{Z} \text{ i } n \in [n_1, n_2]\}$.* ***Diskretni signal*** *je svaka funkcija $x : K \to \mathbb{R}$.*

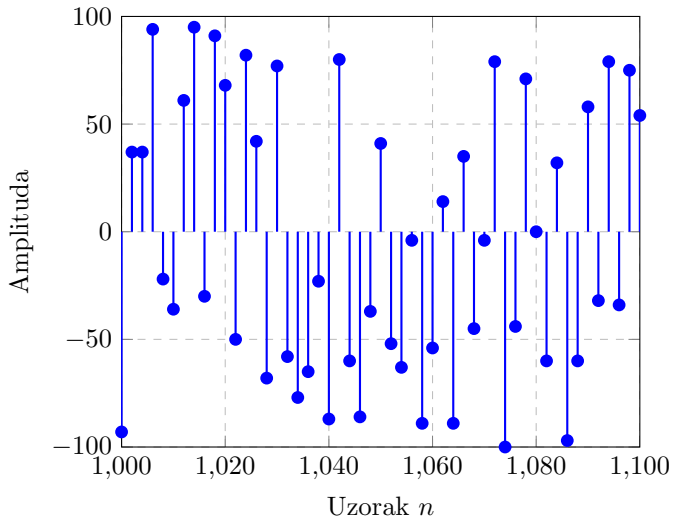
Vrijednost signala u nekoj točki naziva se **amplituda**, dok varijacije amplitude nazivamo val (engl. *waveform*). Primjeri prikaza kontinuiranog i diskretnog signala mogu se vidjeti na slikama 1.1 i 1.2.



Slika 1.1: Prikaz kontinuiranog signala

Diskretni signal se često generira direktno iz kontinuiranog signala procesom **uzorkovanja** (engl. *sampling*) i tada se period T naziva **period uzorkovanja** (engl. *sampling rate*). Također, definiramo i frekvenciju uzorkovanja $f_s = \frac{1}{T}$.

U ovom radu promatrat ćemo upravo takve signale. Točnije, signal kojeg ćemo promatrati je, zapravo, uzorak fiksne veličine nekog kontinuiranog signala dobiven procesom uzorkovanja. Budući da će f_s, odnosno T, biti poznati, uzorak ćemo reprezentirati konačnim nizom realnih brojeva, kojeg ćemo, u skladu sa standardnom



Slika 1.2: Prikaz diskretnog signala pomoću peteljkastog prikaza (engl. *stem plot*)

notacijom u digitalnoj obradi signala, označavati s $\{x[n]\}$. U nekim ćemo slučajevima uzorak formalno promatrati kao vektor iz prostora $\mathbb{R}^N$ ili $\mathbb{C}^N$, gdje je N veličina uzorka. Također, u nekim situacijama nećemo raditi razliku između diskretnog signala i uzorka te ćemo za oba pojma koristiti izraz diskretni signal. Tako definiran diskretni signal trivijalno proširujemo do beskonačnog niza, na način da sve vrijednosti koje nisu definirane postavimo na 0.

Za kraj ovog poglavlja dajemo definiciju sinusoidalnog signala.

Definicija 1.1.3. *Neka su* $t_1, t_2 \in \mathbb{R}$. ***Sinusoidalni signal*** *je signal* $x : [t_1, t_2] \to \mathbb{R}$, *gdje je*

$$x(t) = A\sin(2\pi ft + \phi) = A\sin(\omega t + \phi),$$

pri čemu su:

- A – *realni broj koji označava amplitudu, najveću devijaciju funkcije od* 0
- f – *realni broj koji označava frekvenciju, broj ciklusa koji se pojavljuju u svakoj jedinici vremena (sekunda)*
- $\omega = 2\pi f$ – *realni broj koji označava kutnu frekvencija*
- ϕ – *realni broj koji označava fazu, oscilaciju u* $t = 0$.

Diskretni sinusoidalni signal definira se potpuno analogno na diskretnoj domeni, uz poznati realni T kao u definiciji diskretnog signala.

1.2 Reprezentacija glazbenog signala

U ovom poglavlju idejno opisujemo jednu od formalnih reprezentacija glazbenog signala, odnosno, zvuka. Glazbeni signal najčešće se reprezentira sinusoidalnim modelom koji zvuk aproksimira sumom M sinusoidalnih signala. U nastavku rada smatrat ćemo da su svi signali koje promatramo upravo takvog tipa. Svaki od pojedinih sinusoidalnih signala (sinusoida) naziva se parcijalna komponenta signala (engl. *partial*) s pripadajućom amplitudom i frekvencijom. Strogo formalno, ako promatramo diskretni glazbeni signal $\{x[n]\}$, tada smatramo da je

$$x[n] = \sum_{m=0}^{M-1} A_m \sin(\phi_m(n)) + \epsilon(n),$$

gdje se ϕ_m često promatra na način

$$\phi_m(n) = \int_0^{n/f_s} 2\pi f_m(t)dt + \phi_m(0),$$

pri čemu je f_s frekvencija uzorkovanja, $f_m(t)$ frekvencija u ovisnosti o vremenu, a $\epsilon(n)$ predstavlja šum koji se pojavljuje u signalu. Daljnje detalje ovakve reprezentacije ćemo izostaviti, jer izlaze iz okvira ovog rada, no upućujemo čitatelja na knjigu [12].

1.3 Osnovni pojmovi teorije glazbe

U ovom poglavlju navest ćemo neke osnovne pojmove iz teorije glazbe koje ćemo koristiti u daljnjem radu. Detaljno ćemo objasniti pojam visine tona, koji zauzima centralnu ulogu u problemu kojim se bavimo. Nadalje, detaljnije ćemo opisati pojam harmonije, koji pojašnjava glavni cilj ovog rada, postizanja automatske harmonizacije početnog glazbenog signala. Na kraju poglavlja navodimo jedan primjer koji će dati dobar uvod u problem kojeg obrađujemo te će poslužiti kao most između teorije glazbe i teorije digitalne obrade signala.

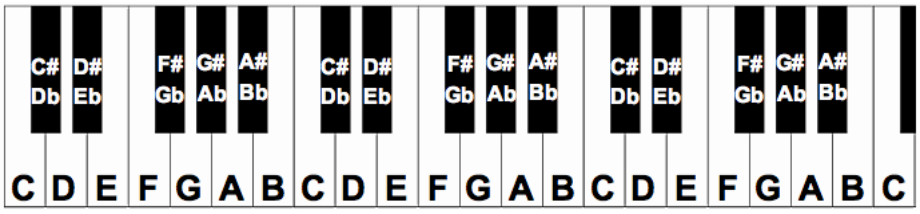
Ton

Svako glazbeno djelo se sastoji od tonova, koji u tom djelu, imaju točno određenu ulogu. Intuitivno, ton označava ono što u nekom trenutku čujemo kad slušamo neko glazbeno djelo. Svaki od tonova koji se pojavljuju u glazbenom djelu ima svoju duljinu, visinu, glasnoću i boju, svojstva koja su direktno vezana uz fizikalna svojstva valova koji su generirani preko glazbenog instrumenta, koji na neki određeni način "proizvodi" zvuk ili, naravno, ljudskog glasa. Čisti ton (engl. *pure tone*) je ton

koji odgovara nekom sinusoidalnom valu koji, neovisno o amplitudi i fazi, ima točno određenu frekvenciju. Čiste tonove promatramo kao osnovne gradivne elemente za kompleksnije valove koji stvaraju glazbu na način kako ju ljudsko uho percipira.

Visina tona (engl. *pitch*) je apstraktni glazbeni pojam koji nam omogućuje poredak tonova u glazbene ljestvice. On je usko vezan uz pojam frekvencije, no ta dva pojma nipošto nisu ekvivalentna. Frekvencija je veličina koja se može fizikalno izmjeriti, dok je visina tona subjektivna percepcija zvučnog vala koja ne može biti egzaktno izmjerena. Visina tona nam zapravo daje neku vrstu uređaja među tonovima. Već smo naveli da je pojam visine tona usko vezan uz pojam frekvencije i to na način da je svaki ton opisan svojom fundamentalnom frekvencijom, najnižom frekvencijom pripadajućeg periodičnog vala koji opisuje taj ton. U duhu našeg razmatranja u prethodnom poglavlju (vidi poglavlje 1.2), fundamentalna frekvencija je najniža frekvencija od svih parcijalnih sinusoidalnih komponenti koji čine glazbeni signal. Pojam usko vezan uz pojam fundamentalne frekvencije je pojam harmonika. Harmonik je svaka frekvencija oblika nf_0, gdje je n pozitivan prirodan broj, a f_0 promatrana fundamentalna frekvencija. Fundamentalna frekvencija je, također, (prvi) harmonik.

Sukladno glazbenoj konvenciji, tonovi se, s obzirom na visinu, dijele u 12 klasa koje označavamo s C, C♯, D, D♯, E, F, F♯, G, G♯, A, B♭ i B[1]. Nadalje, tonovi su raspoređeni u oktave. U svakoj oktavi se nalazi točno 12 tonova, po jedan predstavnik iz svake od klasa, u redoslijedu u kojem su prethodno navedeni. Organizacija tonova u oktave najbolje se vidi na prikazu klavirske klavijature (vidi sliku 1.3).



Slika 1.3: Tri oktave na klavirskoj klavijaturi, svaka oktava počinje tonom C.

Uočimo da su neki tonovi označeni s dvije različite oznake. Razlog tome je da isti tonovi u različitom kontekstu imaju različite uloge (za detalje vidi [17]). S obzirom na oktavu u kojoj se nalaze, oznaci pojedinog tona se pridružuje i redni broj oktave. Na primjer, tonu C u trećoj oktavi pridružuje se oznaka C3. Naglasimo da, nakon tona B u k-toj oktavi, dolazi ton C u oktavi s rednim brojem $(k+1)$. Razmak između dva susjedna tona nazivamo polustepen ili poluton. Dva polustepena čine cijeli stepen ili

[1]U hrvatskoj literaturi ton B se obično označava s H, no radi usklađenosti sa standardnom literaturom na engleskom jeziku u ovom radu koristit ćemo oznaku B.

cijeli ton. U primjenama se obično promatra 7 oktava, tj. cijeli raspon standardne klavirske klavijature.

Prema konvenciji, fundamentalna frekvencija koja služi kao referentna točka za sve ostale frekvencije je frekvencija od 440 Hz (ujedno i frekvencija vala koji proizvodi glazbena vilica koja služi za ugađanje glazbenih instrumenata). Ta frekvencija je fundamentalna frekvencija tona A4. Ako tu frekvenciju označimo s $f_0 = 440$ Hz, tada sve ostale fundamentalne frekvencije računamo po formuli

$$f = 2^{s/12} f_0, \tag{1.1}$$

gdje je $s \in \mathbb{Z}$ broj polutonova za koji je ton, čiju frekvenciju želimo saznati, udaljen od tona A4 (negativni brojevi reprezentiraju pomak "ulijevo", a pozitivni "udesno", vidi [12]). Veća fundamentalna frekvencija odgovara višim tonovima, dok manja fundamentalna frekvencija odgovara dubljim tonovima. Tablica 1.1 daje prikaz fundamentalnih frekvencija svih tonova u 7 oktava. Bitno je uočiti sljedeće: fundamentalna frekvencija tona s oznakom x u k-toj oktavi je dva puta veća od fundamentalne frekvencije tona u $(k+1)$-oj oktavi. To svojstvo će imati bitnu ulogu u razvoju *Pitch Shifting* algoritma.

	C	C♯	D	D♯	E	F	F♯	G	G♯	A	B♭	B
1	32.7	34.6	36.7	38.9	41.2	43.7	46.2	49.0	51.9	55.0	58.3	61.7
2	65.4	69.3	73.4	77.8	82.4	87.3	92.5	98.0	103.8	110.0	116.5	123.5
3	130.8	138.6	146.8	155.6	164.8	174.6	185.0	196.0	207.7	220.0	233.1	246.9
4	261.6	277.2	293.7	311.1	329.6	349.2	370.0	392.0	415.3	440.0	466.2	493.9
5	523.3	554.4	587.3	622.3	659.3	698.5	740.0	784.0	830.6	880.0	932.3	987.8
6	1046.5	1108.7	1174.7	1244.5	1318.5	1396.9	1480.0	1568.0	1661.2	1760.0	1864.7	1975.5
7	2093.0	2217.5	2349.3	2489.0	2637.0	2793.8	2960.0	3136.0	3322.4	3520.0	3729.3	3951.1

Tablica 1.1: Tablica fundamentalnih frekvencija

Sljedeće svojstvo tonova koje navodimo je boja tona. Naime, tonovi iste visine reproducirani na različitim instrumentima ne zvuče jednako. Svaki od instrumenata pojedinom tonu daje boju (engl. *timbre*) karakterističnu za taj instrument. Čisti ton sastoji se samo od fundamentalne frekvencije, dok se tonovi generirani glazbenim instrumentima sastoje od fundamentalne frekvencije s određenom amplitudom, ali i od odgovarajućih harmonika. Upravo različite amplitude harmonika daju boju tonu i omogućuju da isti tonovi odsvirani na različitim instrumentima zvuče drugačije (vidi [19]).

Harmonija

Niz tonova točno određene visine nazivamo melodija. To je ono po čemu je neko glazbeno djelo prepoznatljivo i pamtljivo. Akord je skup od tri ili više tonova koji

zvuče istodobno ili ostavljaju dojam suzvučnosti. Dio teorije glazbe koji proučava akorde naziva se harmonija. Pod pojmom harmonije često se smatra i sama gradnja određenog akorda s obzirom na dani početni ton. Harmonija, u tom smislu, početnu melodiju upotpunjuje i daje joj potpuno novi smisao. Ponovno navodimo primjer zborskog pjevanja. Solo pjevač može pjevati samo jednu melodiju. No, zbor može istovremeno pjevati toliko različitih melodija koliko ima pjevača u zboru i pritom stvarati razne harmonije (akorde).

Ključnu ulogu u harmoniji igraju intervali, strogo definirani razmaci među tonovima. Osnovni intervali (unutar jedne oktave) dani su u tablici 1.2 (za detalje vidi[2]).

Interval	Razmak (polutonovi)	Interval	Razmak (polutonovi)
Mala sekunda	1	Čista kvinta	7
Velika sekunda	2	Mala seksta	8
Mala terca	3	Velika seksta	9
Velika terca	4	Mala septima	10
Čista kvarta	5	Velika septima	11
Povećana kvarta	6	Čista oktava	12

Tablica 1.2: Osnovni intervali u glazbi

Intervali mala i velika sekunda, mala i velika septima te povećana kvarta u praksi daju disonantne zvukove te ih u ovom radu nećemo koristiti. Kao što je već navedeno u uvodnom poglavlju, osnovni cilj ovog rada je razvoj algoritma za automatsko stvaranje različitih harmonija u odnosu na početni ulazni signal.

Tonalitet

U nastavku, opisujemo pojam tonaliteta u glazbi. Definiciju tonaliteta izreći ćemo samo okvirno, jer njezino obrazloženje, kao i sve ostale formalne definicije teorije glazbe, nadilaze potrebe ovog rada (pozivamo čitatelja da za sve detalje pogleda knjigu [23]). Tonalitet, ili ključ glazbenog djela, definira se kao grupa tonova (ljestvica) nad kojom je određeno glazbeno djelo nastalo. Tonalitetu se daje ime s obzirom na prvi ton koji se pojavljuje u ljestvici (prvi stupanj koji se naziva tonika).

U radu ćemo promatrati samo *durske* i *molske*[2] tonalitete. Oni se razlikuju po načinu na koji su građeni. I durska i molska ljestvica sastoje se od 8 tonova (stupnjeva), s tim da su 1. i 8. stupanj isti ton (tonika) koji se razlikuje za točno jednu oktavu. Karakteristika durske ljestvice je da ima polustepene između 3. i 4., te 7. i

[2]Pod pojmom molski tonalitet smatramo prirodni molski tonalitet (vidi [23])

8. stupnja. S druge strane, molska ljestvica ima polustepene između 2. i 3., te 5. i 6. stupnja. Svi ostali stupnjevi odvojeni su jednim cijelim stepenom. Na primjer, D dur ljestvica sastoji se od tonova D, E, F♯, G, A, B, C♯, D. Nadalje, D mol ljestvica se sastoji od tonova D, E, F, G, A, Bb, C, D. Uočimo da u prethodnom primjeru zaista vrijede tvrdnje iz definicije durskog, odnosno, molskog tonaliteta. Kažemo da neki ton pripada tonalitetu, ako pripada njegovoj ljestvici.

Tonalitet će imati veliku ulogu u automatskom stvaranju harmonija. Naime, ako još jednom promotrimo tablicu 1.2, uočit ćemo da neki intervali imaju zajednički drugi dio imena. To su sekunde, terce, sekste i septime[3]. Razlog tome je da se oni formalno definiraju kao razmaci između, redom, dva, tri, šest ili sedam tonova. Razliku im daje način na koji su oni građeni u smislu cijelih stepena ili polustepena. Na primjer, terca formalno u glazbi predstavlja razmak između tri tona. Mala terca je građena od jednog cijelog stepena i jednog polustepena, dok je velika terca građena od dva cijela stepena.

Prilikom stvaranja harmonija mi nećemo eksplicitno navoditi da želimo da na početnom signalu bude izgrađen interval male ili velike terce, jer tako ne bismo nužno dobili harmonije koje su uhu ugodne. Takav bi pristup često rezultirao disonantnim tonovima. Ono što ćemo navoditi je da želimo da se nad početnim signalom izgradi terca. O tome, da li će za pojedini ton biti izgrađena velika ili mala terca, brine tonalitet u kojem se nalazimo. Naime, ako smo u C dur tonalitetu koji se sastoji od tonova C, D, E, F, G, A, B i C i želimo izgraditi tercu na tonu C, to će biti ton E, što je interval velika terca. S druge strane, ako želimo izgraditi tercu na tonu A, to će biti ton C, što je interval mala terca.

Još jednom napominjemo da je ovaj prikaz osnovnih pojmova teorije glazbe neformalan te je naveden s ciljem da se dobije potrebna intuicija.

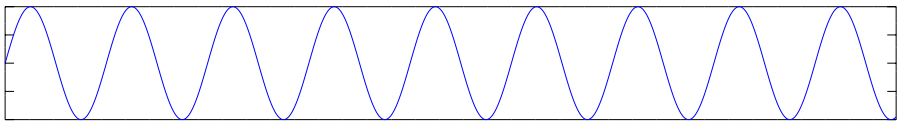
Za kraj ovog poglavlja, donosimo primjer harmonije kroz konstrukciju intervala male terce za neki početni signal (vidi primjer 1.3.1). Radi jednostavnosti prikaza, u ovom (i samo ovom) primjeru promatrat ćemo kontinuirane signale.

Primjer 1.3.1. *Neka je dan signal koji predstavlja čisti ton A4, sinusoidalni signal s frekvencijom* 440 Hz*, bez pomaka u fazi, s amplitudom* 1*. Želimo nad tim početnim tonom "izgraditi" interval malu tercu. Također, želimo da nakon tog imamo signal koji predstavlja oba tona. Mala terca na tonu A4 je ton koji je udaljen od tog tona točno tri polustepena (prema "gore"). To je ton C5 s pripadajućom fundamentalnom frekvencijom koja iznosi* 523.3 Hz*. Zaključujemo da je za "gradnju" intervala male terce na tonu A4 potrebno generirati sinusoidalni signal s frekvencijom od* 523.3 Hz*.*

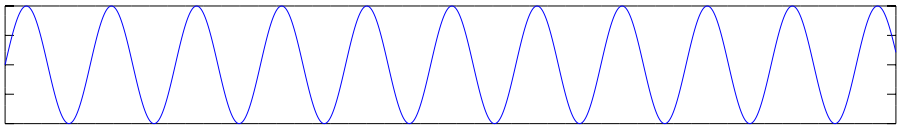
Oba sinusoidalna signala moraju imati identična svojstva, do na frekvenciju. Iz tog razloga je amplituda novog signala jednaka 1*, te, također, nema pomaka u fazi.*

[3]Povećana kvarta je poseban slučaj kojim se nećemo baviti u ovom radu (za detalje vidi [23]).

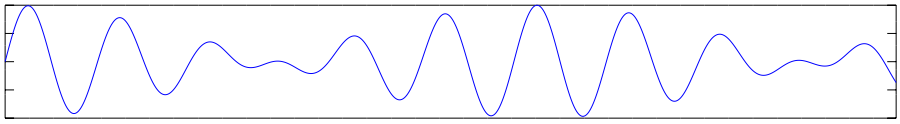
Da bismo dobili signal koji reprezentira oba tona istovremeno, u skladu s prethodnim razmatranjima, potrebno je zbrojiti ta dva sinusoidalna signala. Svaki od inicijalnih signala predstavlja jednu parcijalnu komponentu dobivenog glazbenog signala, koji predstavlja interval male terce na tonu A4 (vidi slike 1.4, 1.5 i 1.6).



Slika 1.4: Čisti ton A4, $\sin(2\pi \cdot 440\, t)$



Slika 1.5: Čisti ton C5, $\sin(2\pi \cdot 523.3\, t)$



Slika 1.6: Tonovi A4 i C5 istovremeno, interval mala terca, pripadni signal je zbroj $\sin(2\pi \cdot 440\, t) + \sin(2\pi \cdot 523.3\, t)$

Poglavlje 2

Diskretna Fourierova transformacija

U ovom poglavlju detaljnije ćemo se baviti tehnikom koju nazivamo Fourierova transformacija. Ona igra ključnu ulogu u algoritmu, u realnom vremenu, kojeg ćemo implementirati za potrebe harmonizacije glazbenog signala (vidi poglavlje 3). Fourierova transformacija, u primjenama u obradi signala, ima ulogu prijelaza iz vremenske reprezentacije signala (signala kao funkcije vremena) u frekvencijsku reprezentaciju signala (signala kao funkcije frekvencije). U širem smislu, ona nam daje mogućnost da početni signal, kao funkciju vremena, rastavimo na njegove sastavne komponente, odgovarajuće sinusoidne signale, te da saznamo informaciju o njihovoj amplitudi i fazi. Budući da u ovom radu promatramo diskretne signale, koncentrirat ćemo se na diskretnu Fourierovu transformaciju (engl. *Discrete Fourier Transform*), odnosno, kratkoročnu Fourierovu transformaciju (engl. *Short-time Fourier Transform*), koje su diskretni analogoni standardne, kontinuirane Fourierove transformacije iz područja Fourierove analize. Kontinuirane Fourierove transformacije u tom, širem, smislu nećemo promatrati, jer bi to izlazilo iz okvira ovog rada, no pozivamo čitatelja da kao uvod u to široko matematičko područje pogleda u [1].

Za početak, definirat ćemo diskretnu Fourierovu transformaciju te uvesti efikasan algoritam za njezino računanje, kojeg zovemo brza Fourierova transformacija (engl. *Fast Fourier Transform*). Zatim ćemo uvesti pojam kratkoročne Fourierove transformacije te opisati njezinu ulogu u obradi signala.

2.1 Diskretna Fourierova transformacija

U nastavku ćemo definirati diskretnu Fourierovu transformaciju (DFT), jednu od najvažnijih tehnika iz područja digitalne obrade signala. Tom tehnikom diskretni sig-

nal duljine N dijelimo na N parcijalnih frekvencijskih komponenti s odgovarajućom amplitudom i pomakom u fazi. Frekvencije tih frekvencijskih komponenti su uniformno raspoređene po frekvencijskom spektru od 0 do f_s, gdje je f_s frekvencija uzorkovanja promatranog diskretnog signala. Takvu interpretaciju diskretne Fourierove transformacije često ćemo koristiti u daljnjim poglavljima prilikom opisivanja algoritma za harmonizaciju.

Slijedi formalna definicija diskretne Fourierove transformacije za diskretne signale.

Definicija 2.1.1. *Neka je $\{x[n]\}$ neki diskretni signal duljine N.* ***Diskretna Fourierova transformacija*** *(DFT) signala $\{x[n]\}$ je niz od N kompleksnih brojeva danih izrazom*

$$X[k] = \sum_{n}^{N-1} x[n] W_N^{-kn},$$

gdje je $W_N = e^{\frac{2\pi i}{N}}$ i $k = 0, \ldots, N-1$.

Napomenimo da, prema Eulerovoj formuli, vrijedi

$$W_N^{-kn} = e^{-\frac{i2\pi kn}{N}} = \cos\left(\frac{2\pi kn}{N}\right) - i\sin\left(\frac{2\pi kn}{N}\right).$$

Brojeve W_N^k, za $k \in 0, \ldots, N-1$, često nazivamo N-ti korijeni iz jedinice.

Na vrijednost $X[k]$ ćemo se, u nastavku rada, nekoliko puta referirati kao na sadržaj k-tog frekvencijskog spremnika diskretne Fourierove transformacije (engl. *frequency bin*). Za svaki $k \in \{0, \ldots, N-1\}$, $X[k]$ kodira vrijednost amplitude i faze odgovarajuće parcijalne frekvencijske komponente s pripadnom frekvencijom $\frac{k}{N} f_s$. Amplituda je, zapravo, modul kompleksnog broja $X[k]$,

$$A[k] = \sqrt{\mathrm{Re}(X[k])^2 + \mathrm{Im}(X[k])^2}.$$

Faza pripadne sinusoidalne frekvencijske komponente računa se po formuli

$$\phi[k] = \mathrm{atan2}(\,\mathrm{Im}(X[k]), \mathrm{Re}(X[k])\,) \in [-\pi, \pi].$$

Napomenimo da nećemo detaljnije ulaziti u samu definiciju i svojstva diskretne Fourierove transformacije, jer bi to izlazilo iz okvira ovog rada. Pozivamo čitatelja da za sve detalje prouči knjigu [18].

DFT kao linearan operator

U nastavku ovog poglavlja promatrat ćemo vektorski prostor $\mathbb{C}^N$ te ćemo definirati diskretnu Fourierovu transformaciju kao linearan operator na tom prostoru. Uočimo

da takva interpretacija ima smisla sa stajališta diskretnih signala, jer se svaki diskretni signal duljine N može promatrati kao N-dimenzionalni vektor iz prostora $\mathbb{C}^N$. Činjenica da je diskretna Fourierova transformacija linearan operator bit će izuzetno važna prilikom implementacije programskog rješenja kojeg opisujemo u poglavlju 4.

Napomena 2.1.2. *Radi jednostavnosti i preglednosti koristimo indeksiranje od* 0*, a ne od* 1*, kako je to karakteristično za linearnu algebru. U skladu s tim, koeficijente vektora $x \in \mathbb{C}^N$ indeksiramo vrijednostima iz skupa $\{0, \ldots, N-1\}$. Konkretnije, $x = (x_0, \ldots, x_{N-1})^\tau$.*

Definicija 2.1.3. *Neka je $N \in \mathbb{N}$ i neka je $a = (a_0, a_1, \ldots, a_{N-1})^\tau \in \mathbb{C}^N$. Kažemo da je vektor $y \in \mathbb{C}^N$* ***diskretna Fourierova transformacija*** *vektora a, u oznaci $y = \mathrm{DFT}_N(a)$, ako vrijedi*

$$y_k = \sum_{j=0}^{N-1} a_j W_N^{-kj}, \text{ za svaki } k \in \{0, \ldots, N-1\},$$

gdje je $W_N = e^{\frac{2\pi i}{N}}$.

Propozicija 2.1.4 pokazuje da je DFT_N linearan operator, za svaki $N \in \mathbb{N}$. Iako je dokaz očit, navodimo ga radi potpunosti.

Propozicija 2.1.4. *Neka je $N \in \mathbb{N}$. Onda je $\mathrm{DFT}_N : \mathbb{C}^N \to \mathbb{C}^N$ linearan operator.*

Dokaz. Neka su $a, b \in \mathbb{C}^N$ i neka su $\alpha, \beta \in \mathbb{C}$. Za svaki $i \in \{0, \ldots, N-1\}$ vrijedi:

$$\begin{aligned}(\mathrm{DFT}_N(\alpha a + \beta b))_i &= \sum_{j=0}^{N-1} (\alpha a + \beta b)_j W_N^{-kj} = \sum_{j=0}^{N-1} \alpha a_j W_N^{-kj} + \sum_{j=0}^{N-1} \beta b_j W_N^{-kj} \\ &= \alpha \sum_{j=0}^{N-1} a_j W_N^{-kj} + \beta \sum_{j=0}^{N-1} b_j W_N^{-kj} = \alpha(\mathrm{DFT}_N(a))_i + \beta(\mathrm{DFT}_N(b))_i.\end{aligned}$$

Time je tvrdnja dokazana. □

Matrični zapis operatora DFT_N je matrica $V_N \in \mathbb{C}^{N \times N}$ oblika

$$\begin{bmatrix} 1 & 1 & 1 & 1 & \ldots & 1 \\ 1 & W_N^{-1} & W_N^{-2} & W_N^{-3} & \ldots & W_N^{-(N-1)} \\ 1 & W_N^{-2} & W_N^{-4} & W_N^{-6} & \ldots & W_N^{-2(N-1)} \\ 1 & W_N^{-3} & W_N^{-6} & W_N^{-9} & \ldots & W_N^{-3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{-(N-1)} & W_N^{-2(N-1)} & W_N^{-3(N-1)} & \ldots & W_N^{-(N-1)(N-1)} \end{bmatrix},$$

takozvana Vandermondova matrica, koja sadrži odgovarajuće potencije od W_N^{-1}. Točnije, element na mjestu (i, j) matrice V_N jednak je W_N^{-ij}, za svaki $i, j \in \{0, \ldots, N-1\}$. Poznato je da je determinanta Vandermondove matrice (evaluirana u potencijama od W_N^{-1}) dana izrazom (vidi, na primjer [14])

$$\det(V_N) = \prod_{0 \leq i < j \leq N-1} (W_N^{-i} - W_N^{-j}).$$

Kako su svi W_N^{-k} različiti, za $k \in \{0, \ldots N-1\}$, zaključujemo da je $\det(V_N) \neq 0$. Iz tog slijedi da je V_N invertibilna.

Sada se lako pokaže da, ako je $y = (y_0, \ldots, y_{N-1})^\tau$ neki vektor iz prostora $\mathbb{C}^N$, tada je njegova **inverzna** Fourierova transformacija vektor $x \in C^N$, dan izrazom

$$x_k = \frac{1}{N} \sum_{j=0}^{N-1} y_j W_N^{kj},$$

za svaki $k = 0, \ldots, N-1$.

2.2 Brza Fourierova transformacija

U ovom poglavlju predstavit ćemo algoritam koji se u literaturi naziva brza Fourierova transformacija (engl. *Fast Fourier Transform*, ili kratko FFT), kojim se efikasno izračunava diskretna Fourierova transformacija diskretnog signala duljine N, gdje je N potencija broja 2. Radi jednostavnije notacije, u ovom poglavlju ćemo ponovno diskretni signal promatrati kao vektor iz $\mathbb{C}^N$.

Uočimo da direktni algoritam za računanje diskretne Fourierova transformacije vektora dimenzije N ima kvadratnu složenost, budući da nam je za svaku od N komponenti rezultantnog vektora potrebno N množenja i N zbrajanja. U mnogim primjenama, između ostalog i u ovom radu, zahtijeva se efikasnija implementacija diskretne Fourierove transformacije, budući da je ona osnova za mnoge algoritme iz sfere digitalne obrade signala, koji su u pravilu vremenski ograničeni, tj. zahtijevaju izvršavanje u realnom vremenu.

Metoda koju ćemo predstaviti u ovom poglavlju bazira se na specijalnim svojstvima N-tih korijena iz jedinice i koristi metodu *podijeli pa vladaj* (engl. *divide-and-conquer*). Najprije ćemo opisati osnovni rekurzivni algoritam, a zatim ćemo predstaviti efikasnu iterativnu implementaciju, koju ćemo koristiti u programskom rješenju opisanom u daljnjim poglavljima.

Rekurzivni FFT

Neka je N potencija od 2 i neka je dan vektor $a = (a_0, \ldots, a_{N-1})$. Prema definiciji 2.1.3, njegova DFT je vektor $y = (y_0, \ldots y_{N-1})$, definiran s

$$y_k = \sum_{j=0}^{N-1} a_j W_N^{-kj}. \tag{2.1}$$

Uočimo da izraz (2.1) možemo promatrati kao na izvrednjavanje polinoma

$$A(x) = \sum_{j=0}^{N-1} a_j x^j$$

u točkama W_N^{-k}, za $k \in \{0, \ldots N-1\}$.

Korištenjem metode podijeli pa vladaj možemo definirati dva nova polinoma, $A_0(x)$ i $A_1(x)$, oba stupnja $\frac{N}{2} - 1$,

$$A_0(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{N-2} x^{\frac{N}{2}-1},$$
$$A_1(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{N-1} x^{\frac{N}{2}-1}.$$

Uočimo da A_0 sadrži sve koeficijente polinoma A s parnim indeksima, dok A_1 sadrži sve koeficijente s neparnim indeksima. Tada $A(x)$ lako izračunamo na sljedeći način:

$$A(x) = A_0(x^2) + xA_1(x^2). \tag{2.2}$$

Na taj način smo početni problem evaluiranja polinoma $A(x)$ u W_N^{-k} rastavili na evaluiranje dva polinoma, $A_0(x)$ i $A_1(x)$ u točkama $(W_N^0)^2, (W_N^{-1})^2, \ldots, (W_N^{-(N-1)})^2$ te primjenu izraza (2.2). Lako se pokaže da su kvadrati tih N N-tih korijena iz jedinice jednaki $\frac{N}{2}$ različitih $\frac{N}{2}$-tih korijena iz jedinice (vidi [4]). Dakle, nove polinome, zapravo, evaluiramo u točkama $(W_N^0), (W_N^{-1}), \ldots, (W_N^{-\frac{(N-1)}{2}})$. Time smo početni problem određivanja DFT N-dimenzionalnog vektora podijelili na problem određivanja DFT dva $\frac{N}{2}$-dimenzionalna vektora.

Iz gornjeg razmatranja lako slijedi rekurzivan algoritam za računanje DFT, poznat kao rekurzivni FFT. Pseudo-kod algoritma dan je algoritmom 1.

Uočimo da izraz iz 12. retka algoritma,

$$y_{k+\frac{N}{2}} = y_k^0 - W y_k^1,$$

slijedi iz činjenice da vrijedi

$$W_N^{k+\frac{N}{2}} = -W_N^k,$$

Algoritam 1 Rekurzivni FFT algoritam

```
1: function recursiveFFT(a, N)
2:     if n = 1 then return a
3:     end if
4:     W_N = e^{-2πi/N}
5:     W = 1
6:     a^0 = (a_0, a_2, ..., a_{N-2})
7:     a^1 = (a_1, a_3, ..., a_{N-1})
8:     y^0 = recursiveFFT(a^0)
9:     y^1 = recursiveFFT(a^1)
10:    for k = 0 to n/2 − 1 do
11:        y_k = y_k^0 + W y_k^1
12:        y_{k+N/2} = y_k^0 − W y_k^1
13:        W = W W_N
14:    end for
15:    return y
16: end function
```

što se, također, lako pokaže (vidi [4]).

Operaciju oblika

$$y_k = y_k^0 + W y_k^1, \qquad y_{k+N/2} = y_k^0 - W y_k^1, \tag{2.3}$$

kakvu vidimo u 11. i 12. retku algoritma, gdje najprije broj pomnožimo nekim drugim brojem pa rezultat dodamo i oduzmemo od nekog trećeg broja, nazivamo *leptir* operacija.

Nadalje, uočimo da je vrlo jednostavno modificirati opisani algoritam da računa inverznu DFT. Naime, u opisanom pseudo-kodu, dovoljno je zamijeniti uloge a i y, inicijalno postaviti $W_N = e^{\frac{2\pi i}{N}}$ te, na kraju, cijeli vektor skalirati faktorom $\frac{1}{N}$.

Uočimo da vrijeme izvršavanja opisanog algoritma zadovoljava rekurziju

$$T(n) = 2T\left(\frac{n}{2}\right) + \mathcal{O}(n).$$

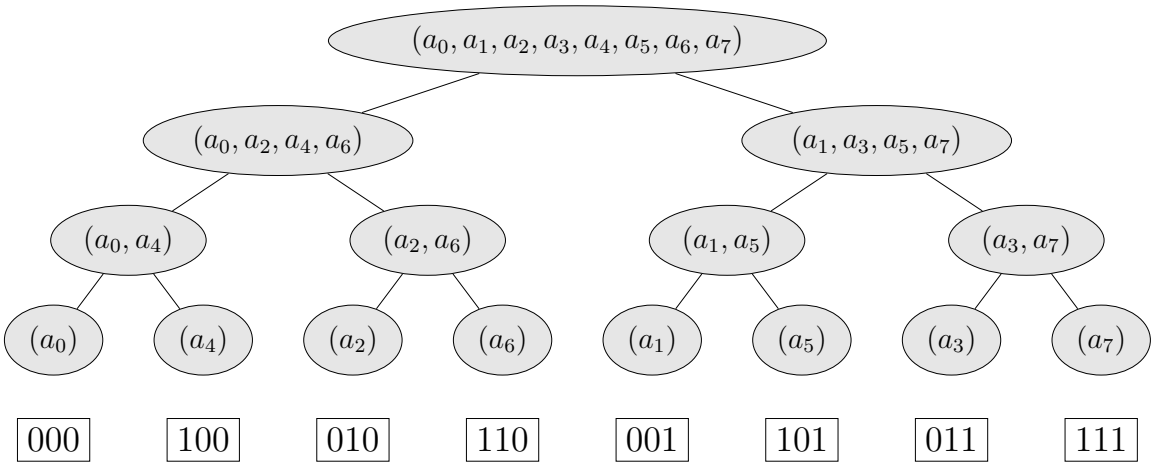
Iz tog zaključujemo da je vremenska složenost rekurzivnog FFT algoritma $\mathcal{O}(n \log n)$. Štoviše, vrijedi da je vremenska složenost algoritma $\Theta(n \log n)$.

Iterativni FFT

U nastavku ćemo opisati iterativnu verziju opisanog rekurzivnog algoritma, koju ćemo koristiti u daljnjem radu. Napomenimo da će finalna verzija algoritma imati

istu asimptotsku ocjenu složenosti $\mathcal{O}(n \log n)$, no konstanta uz izraz $n \log n$ bit će manja. Također, riješit ćemo se neefikasnih rekurzivnih poziva. Ono što je najvažnije, implementaciju možemo napraviti koristeći samo jedan pomoćni vektor, u koji ćemo spremiti završni rezultat, što kod rekurzivne implementacije nije bilo moguće.

Promotrimo najprije na koji način se ponaša opisani rekurzivni algoritam. Stablo rekurzije za $N = 8$ prikazano je na slici 2.1



Slika 2.1: Stablo rekurzije rekurzirvnog FFT algoritma

Uočimo da, ako inicijalno uspijemo poredati elemente vektora u redoslijedu u kojem su oni poredani u listove na slici 2.1 slijeva nadesno, možemo simulirati izvršavanje rekurzivnog FFT algoritma. U prvom koraku, nakon mijenjanja poretka elemenata vektora, uzimamo elemente u paru, kako se sekvencijalno pojavljuju u novom vektoru, izračunamo DFT od svakog para koristeći jednu *leptir operaciju* (vidi izraz (2.3)) te zamijenimo par s dobivenom transformacijom. Nakon prvog koraka imamo $\frac{N}{2}$ 2-članih transformacija. Sada uzmemo tih $\frac{N}{2}$ transformacija u parovima, kako se sekvencijalno pojavljuju, te računamo njihove DFT koristeći dvije *leptir* operacije. Dobili smo $\frac{N}{4}$ novih transformacija od kojih svaka sadrži 4 elementa. Analogno nastavljamo dalje, sve dok ne preostanu dvije transformacije s $\frac{N}{2}$ elementa, za koje nam je potrebno $\frac{N}{2}$ *leptir* operacija za izračunati DFT originalnog vektora.

Koristeći gornja razmatranja, uvodimo novu varijablu kojom brojimo razine stabla koje prolazimo. Budući da je stablo rekurzije rekurzivnog FFT algoritma potpuno binarno stablo s N listova, visina mu je $\log_2(N)$ (cijeli broj, budući da je N potencija broja 2). Za svaku od razina, potrebno je napraviti opisane *leptir* operacije. U polje A spremamo rezultate transformacija na pojedinim razinama. Nakon završetka rada algoritma, A sadrži rezultat, DFT originalnog vektora a. Pseudo-kod algoritma dan je algoritmom 2 (za tehničke detalje vidi [4]).

Algoritam 2 Iterativni FFT algoritam

```
1: function iterativeFFT(a, N)
2:     bitReverseCopy(a, A);
3:     for s = 1 to log_2 N do
4:         m = 2^s
5:         W_N = e^{-2πi/N}
6:         W = 1
7:         for j = 0 to m/2 − 1 do
8:             for k = j to N − 1 by m do
9:                 t = WA[k + m/2]
10:                u = A[k]
11:                A[k] = u + t
12:                A[k + m/2] = u − t
13:            end for
14:            W = WW_N
15:        end for
16:    end for
17:    return A
18: end function
```

U algoritmu 2, funkcija *bitReverseCopy*() zadužena je za permutaciju elemenata originalnog vektora a u redoslijed pogodan za rad iterativnog algoritma. Vratimo se ponovno na sliku 2.1. Uočimo da se listovi pojavljuju u redoslijedu 0, 4, 2, 6, 1, 5, 3, 7. Ako promotrimo taj niz u binarnom zapisu, dobivamo niz 000, 100, 010, 110, 001, 101, 011, 111. Uočimo da ako u dobivenom nizu sve binarne znamenke zapišemo u obrnutom redoslijedu (*bit reverse* operacija), novi niz je oblika 000, 001, 010, 011, 100, 101, 110, 111, što upravo odgovara početnom redoslijedu u inicijalnom vektoru. To vrijedi i općenito. Naime, budući da u svakom koraku rekurzivnog algoritma dijelimo koeficijente promatranog polinoma na one s parnim indeksom i na one s neparnim indeksom, uvijek će oni s reprezentativnim bitom 0 otići ulijevo, a oni sa reprezentativnim bitom 1 udesno.

Pokažimo još da je za operaciju bitReverseCopy() potrebno $\mathcal{O}(n \log n)$ vremena u veličini vektora N. Naime, za svaki od N indeksa potrebno je napraviti *bit reverse* operaciju, za koju moramo jednom proći po svim znamenkama promatranog indeksa (uključujući eventualne vodeće nule). Veličina vektora nad kojim radimo DFT uglavnom je poznata unaprijed (u našem radu će to biti veličina okvira promatranog diskretnog signala kojeg procesiramo) pa se indeksi mogu permutirati samo jednom, na početku izvršavanja programa.

Iz danog pseudo-koda lako je izvesti formulu za broj operacija u iterativnoj verziji FFT algoritma (vidi [4]). Na taj način (vidi [4]), zaključujemo da je vremenska složenost iterativnog FFT algoritma $\mathcal{O}(n \log n)$. Štoviše, može se pokazati da je vremenska složenost iterativnog FFT algoritma $\Theta(n \log n)$.

2.3 Kratkoročna Fourierova transformacija

U prethodnom poglavlju o diskretnoj Fourierovoj transformaciji, pokazali smo na koji način svaki diskretni signal, odnosno uzorak, možemo prikazati kao sumu parcijalnih frekvencijskih komponenti s odgovarajućim amplitudama i pomacima u fazi. Uočimo kako se glazbeni signali mijenjaju u odnosu na vremenski interval koji promatramo. Štoviše, moramo uzeti u obzir da stacionarni signali (signali koji se s vremenom ne mijenjaju), s gledišta glazbe, nisu pretjerano zanimljivi. Kad bolje promotrimo glazbene signale, uočit ćemo da se oni prilično intenzivno mijenjaju u vremenu, stvarajući razne melodije. Ako uzmemo dovoljno malen okvir, koji reprezentira signal u tek nekoliko milisekundi, zaključujemo da je unutar tog vremenskog intervala signal zapravo stacionaran, budući da predstavlja samo jedan ton (ili više tonova ako se radi o već harmoniziranom signalu). Iz tog razloga, glazbene signale, slično kao i ljudski govor, stavljamo u grupu signala koje nazivamo kratkoročno stacionarni signali.

Iz prethodnih razmatranja zaključujemo kako nema smisla promatrati diskretnu Fourierovu transformaciju na cijelom uzorku promatranog signala, jer se glazbeni signal tijekom vremena intenzivno mijenja. Budući da se u cijelom uzorku pojavljuje mnogo različitih frekvencija, iz rezultata diskretne Fourierove transformacije nemoguće je izvući neke relevantne zaključke. No, ako signal podijelimo u manje okvire (prozore) te za svaki od njih odredimo njihovu diskretnu Fourierovu transformaciju, dobit ćemo informaciju o parcijalnim frekvencijskim komponentama za svaki od tih okvira. Takav postupak nazivamo kratkoročna Fourierova transformacija (engl. *Short-time Fourier Transform*), kratko STFT.

Za početak, definirat ćemo pojam prozorske funkcije.

Definicija 2.3.1. *Neka je N neki prirodan broj. Za funkciju $w : \mathbb{Z} \to \mathbb{R}$ kažemo da je* ***prozorska funkcija*** *ili funkcija prozora ako vrijedi da je $w(k) = 0$, za svaki $k \in \mathbb{Z}$ za kojeg vrijedi da $k \notin [0, N]$.*

Prozorska funkcija je, zapravo, težinska funkcija kojom određujemo prozor, odnosno okvir, duljine N, na kojem promatramo inicijalni signal.

U nastavku donosimo formalnu definiciju kratkoročne Fourierove transformacije.

Definicija 2.3.2. *Neka je $\{x[n]\}$ neki diskretni signal (odnosno, njegovo beskonačno proširenje) te neka je N prirodan broj. Neka je w neka prozorska funkcija.* ***Kratkoročna Fourierova transformacija*** *(STFT) signala $\{x[n]\}$ s indeksom m je dana izrazom*

$$X(m,k) = \sum_{n=-\infty}^{+\infty} x[n]w[n-m]W_N^{-kn},$$

za $k = 0, 1, \ldots, N-1$ i $W_N = \frac{2\pi i}{N}$.

$X(m,k)$ iz definicije STFT interpretira se kao niz od N kompleksnih brojeva koji nose informaciju o amplitudi i fazi pojedine parcijalne frekvencijske komponente, određene frekvencijom $\frac{k}{N}f_s$, gdje je f_s frekvencija uzorkovanja. Indeks m određuje pomak signala u odnosu na prozor, tj. određuje poziciju prozora u promatranom vremenskom trenutku u odnosu na glavni signal. Dakle, indeks m u širem smislu možemo promatrati kao indeks koji određuje vremenski trenutak u kojem promatramo signal.

U skladu sa standradnom notacijom u digitalnoj obradi signala, uglavnom ćemo za vrijednost $X(m,k)$ koristiti oznaku $X_m[k]$.

S odgovarajućim odabirom funkcije w (za detalje vidi [16]) postiže se da originalni niz $\{x[n]\}$ u potpunosti može biti konstruiran izrazom

$$x[n] = \sum_{m=-\infty}^{+\infty} \frac{1}{N} \sum_{k=0}^{N-1} X(m,k)W_N^{kn}.$$

To je takozvana *Overlap-add* metoda, koja se izuzetno često koristi u praksi (vidi [16]). Uočimo da je izraz

$$\frac{1}{N} \sum_{k=0}^{N-1} X(m,k)W_N^{kn},$$

zapravo, inverzna diskretna Fourierova transformacija niza $X(k,m)$.

Za detaljniji uvid u kratkoročnu Fourierovu transformaciju upućujemo čitatelja na knjigu [22]. Napomenimo još da je suma u definiciji STFT, u praksi, uvijek konačna. Razlog tome je taj da je prozorska funkcija definirana na način da je različita od nule samo na konačnom skupu vrijednosti. Njezina uloga je, upravo, podijeliti polazni signal na okvire duljine N te raditi transformacije nad tim segmentima.

Hannov prozor

Prozorske funkcije, osim što određuju okvir, imaju i neka druga korisna svojstva. Naime, prilikom računanja diskretne Fourierove transformacije nekog konačnog signala, implicitno periodički proširujemo početni konačni signal (vidi [18]). U tom

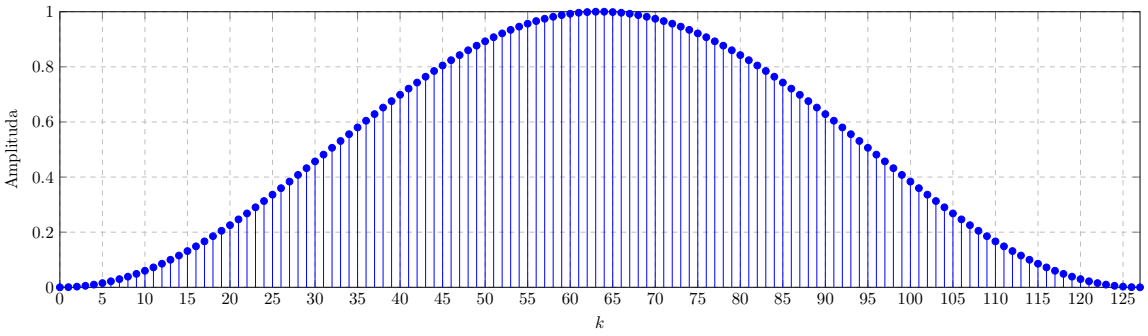
slučaju, ako se početna i završna vrijednost ne poklapaju, pojavljuju se takozvani diskontinuiteti u modificiranom signalu, koji obično rezultiraju detekcijom visokih frekvencija koje se inicijalno ne pojavljuju u originalnom signalu te više nije moguće rekonstruirati originalni signal. Iz tog razloga se u digitalnoj obradi signala često upotrebljavaju prozorske funkcije koje imaju "zvonoliki" oblik, da bi se ta pojava smanjila (za detalje vidi [10]).

Definirat ćemo jednu specijalnu prozorsku funkciju, poznatu pod nazivom Hannov prozor (engl. *Hann window* ili *Hanning window*), koja zadovoljava svojstvo opisano u prethodnom odlomku.

Definicija 2.3.3. *Neka je N prirodan broj.* ***Hannov prozor*** *je funkcija w, gdje je $w : \mathbb{Z} \to \mathbb{R}$ definirana s*

$$w(n) = \begin{cases} \frac{1}{2}\left[1 - \cos\left(\frac{2\pi n}{N-1}\right)\right], & \text{ako je } 0 \leq n \leq N, \\ 0, & \text{inače.} \end{cases}$$

Na slici 2.2 dajemo peteljkasti graf Hannovog prozora za $N = 128$.



Slika 2.2: Hannov prozor za $N = 128$

Poglavlje 3

Pitch Shifting – opis algoritma

U ovom poglavlju opisat ćemo algoritam kojim postižemo efekt harmonizacije u realnom vremenu. Algoritmi tog tipa se grupnim imenom nazivaju *Pitch Shifting* algoritmi.

3.1 Uvod u *Pitch Shifting*

Pitch Shifting algoritmi su algoritmi koji mijenjaju visinu tona početnog signala. Algoritam tog tipa kao ulaz prima neki diskretni signal, najčešće fragmentiran u segmente fiksne veličine. Signal se može sekvencijalno čitati iz neke datoteke (na primjer, u *.wav* formatu[1]) ili direktno primati s nekog audio uređaja (na primjer, računalna zvučna kartica). Također, algoritam kao ulaz mora primiti i interval za koji želimo pomaknuti polazni ton te smjer pomaka. Uz to, potrebna nam je i informacija o tonalitetu, koja predstavlja kontekst u kojem želimo provoditi mijenjanje visine tona. Informacija o tonalitetu sastoji se od početnog tona tonaliteta te zastavice radi li se o durskom ili molskom tonalitetu. Kao izlaz, algoritam na neki audio izlaz (na primjer, računalna zvučna kartica) mora poslati harmonizirani signal. To znači da želimo da rezultat algoritma sadrži i početni ton i njegovu modificiranu verziju te da, na taj način, tvori željenu harmoniju.

Osnovna pretpostavka je da u svakom trenutku početni signal označava točno jedan ton. Naime, ako se početni signal sastoji od više različitih tonova, tada je ulazni signal već harmoniziran, pa harmonizaciju na način kako to opisano u poglavlju 1.3 nema smisla promatrati.

Bitno je napomenuti da ta pretpostavka u praksi nije nužna. Algoritam koji opisujemo moći će procesirati i signale koji reprezentiraju više tonova odsviranih

[1] *Waveform Audio File Format*, vidi [8] ili [11]

odjednom. Prilikom harmonizacije nekog signala, koji se sastoji od više tonova, kao reprezentativnu parcijalnu frekvencijsku komponentu uzet ćemo onu s najvišom amplitudom.

Prije nego što krenemo u analizu algoritma po koracima, pojasnit ćemo na što točno mislimo kad kažemo da algoritam mora raditi u realnom vremenu. To znači da se tijekom rada algoritma ne smije desiti prekid signala ili određena latencija, koja je uzrokovana predugim vremenom obrade samog signala. Poznato je da svi digitalni audio uređaji procesiraju digitalni zvučni signal s nekom frekvencijom uzorkovanja f_s. To znači da, ako na zvučnu karticu pošaljemo segment uzorka veličine L, imamo točno L/f_s sekundi vremena da u potpunosti procesiramo sljedeći segment, prije nego se prvi segment u potpunosti obradi, tj. u slučaju zvučne kartice, reproducira. Ako je potrebno vrijeme obrade veće od L/f_s, tada se pojavljuju "rupe" u signalu, što, budući da obrađujemo glazbeni signal, definitivno želimo izbjeći. Zaključujemo da je prilikom konstrukcije algoritma bitna njegova efikasnost, da bi se zadovoljio uvjet rada u realnom vremenu.

Pitch Shifting algoritam kojeg opisujemo bazira se na *Phase Vocoder* algoritmu, koji je jedan od standardnih algoritama u digitalnoj obradi signala.

3.2 Phase Vocoder

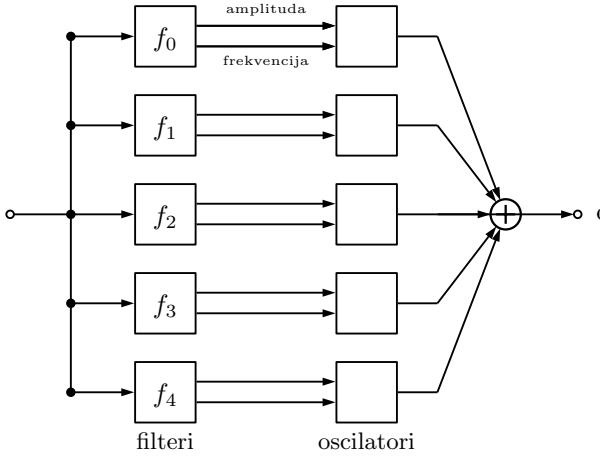
Phase Vocoder je jedna od najpoznatijih i najčešće korištenih tehnika digitalne obrade signala. Povijesno, algoritam proizlazi iz raznih tehnika istraživanih početkom druge polovice 20. stoljeća, koje su primarno razvijane s ciljem kodiranja ljudskog govora, odnosno, ljudskog glasa. Sama riječ *Vocoder* je upravo izvedenica od engleskih riječi *voice*, što znači "glas", te *coder*. Sam algoritam je prvi put opisan u radu Flanagana i Goldena iz 1966. godine (vidi [7]).

Phase Vocoder je jedan od algoritama digitalne obrade signala koji spada u skupinu takozvanih analiza-sinteza (engl. *analysis-synthesis*) algoritama. Takvi algoritmi imaju svojstvo da primaju neki ulazni signal te, kao rezultat, vraćaju signal koji je ili identičan ulaznom signalu, ili je njegova modificirana verzija. Implicitna pretpostavka je da se ulazni signal može prikazati matematičkim modelom koji ovisi o vremenu. U kontinuiranom slučaju, ulazni signal je realna funkcija koja ovisi o vremenu, no za potrebe obrade signala, ulazni signal se promatra kao diskretan skup vrijednosti u nekim, točno određenim vremenskim trenucima. Naglasimo da se, u fazi analize, ulazni signal po potrebi modificira ili su iz njega, ovisno o primjeni, generirane neke tražene vrijednosti. Takvi algoritmi se uvelike koriste u primjenama vezanim uz obradu glazbe ili obradu ljudskog govora. Uočimo da problem promjene visine tona možemo relativno jednostavno staviti u kontekst analize i sinteze. Taj signal mora na neki način biti pročitan i reprezentiran u obliku koji je podložan modifikaciji, u

smislu visine tona (rezonirajuće frekvencije), što odgovara fazi analize. Zatim, signal mora biti na odgovarajući način modificiran da bi se ostvarila promjena visine tona. Nakon modifikacije, modificirani signal se treba vratiti u početnu reprezentaciju, što je zadaća faze sinteze.

Filter bank interpretacija

U nastavku opisujemo osnovnu ideju *Phase Vocoder* algoritma, koristeći takozvanu *Filter bank* interpretaciju. Konkretnije, faza analize u *Phase Vocoder* algoritmu može se interpretirati na način da ulazni signal prolazi kroz konačno mnogo pojasnih (engl. *bandpass*) filtera. Izlaz svakog od filtera su amplituda i frekvencija, koje ovise o vremenu. Faza sinteze je zbroj sinusoida, čije amplitude i frekvencije su dobivene direktno iz svakog pojasnog filtera (prolaskom kroz oscilator). Svaki od filtera ima svoju centralnu frekvenciju, koja određuje dio signala koji će biti procesiran tim filterom. Grafički prikaz *Filter bank* reprezentacije prikazan je na slici 3.1.



Slika 3.1: *Filter bank* interpretacija *Phase Vocoder* algoritma

Opisani skup filtera mora zadovoljavati sljedeća svojstva:

1. svi filteri su ekvivalentni, do na centralnu frekvenciju
2. centralne frekvencije su uniformno raspoređene po cijelom spektru, od 0 Hz pa sve do polovice frekvencije uzorkovanja (posljedica Nyquist–Shannonovog teorema koji će biti spomenut u nastavku, u poglavlju 3.4)
3. svakoj pojedinoj frekvencijskoj komponenti dana je jednaka težina u fazi analize.

Naglasimo da broj pojasnih filtera igra izuzetno bitnu ulogu u radu algoritma. Naime, povećanjem broja filtera, profinjujemo skup frekvencija koje algoritam može profiltrirati, odnosno, registrirati.

Za detaljniji prikaz *Phase Vocoder* algoritma, u smislu *Filter bank* interpretacije, te detaljni opis ponašanja svakog pojasnog filtera, upućujemo čitatelja na članak *The Phase Vocoder: A Tutorial* Marka Dolsona (vidi [6]). Ta interpretaciju ima svoju matematičku podlogu u teoriji diskretne obrade signala, gdje je pojam (pojasnog) filtera egzaktno definiran (vidi, na primjer, [25]).

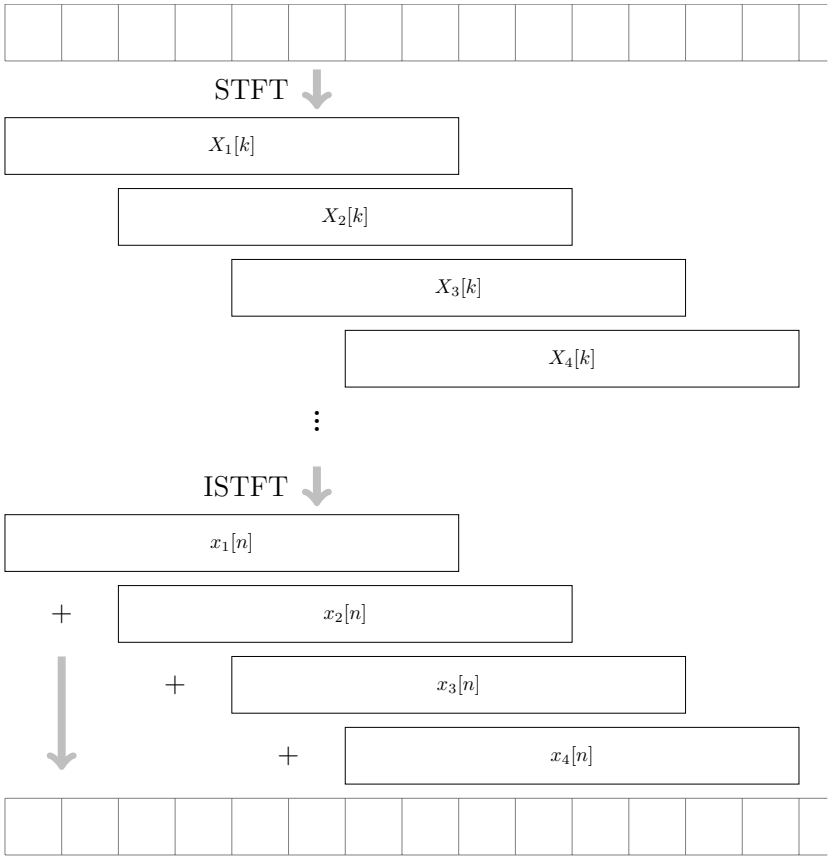
Fourierova interpretacija

U ovom radu, priklonili smo se takozvanoj Fourierovoj interpretaciji *Phase Vocoder* algoritma (vidi [6] i [20]), koja daje drugačiji, no ekvivalentan pogled na sam algoritam. U ovoj interpretaciji, faza analize *Phase Vocoder* algoritma sastoji se od niza preklapajućih kratkoročnih Fourierovih transformacija (vidi poglavlje 2.3) primijenjenih na konačne uzorke (okvire) ulaznog signala. Kao što smo već ranije naveli (vidi poglavlje 2), osnovna uloga Fourierove transformacije je prijelaz iz vremenske reprezentacije signala u frekvencijsku reprezentaciju signala. Na taj način, dobit ćemo informaciju o odgovarajućim amplitudama i fazama, za svaku od parcijalnih frekvencijskih komponenti, za neki vremenski trenutak. Važno je napomenuti da podjela signala na okvire može rezultirati neželjenim pomacima u fazi te je potreban još jedan korak procesiranja, da bi se dobila informacija o pravoj frekvenciji vezanoj uz odgovarajuće frekvencijske spremnike kratkoročne Fourierove transformacije. Korak procesiranja bit će detaljno objašnjen u sljedećem poglavlju (vidi poglavlje 3.4). Faza sinteze je, u ovom slučaju, realizirana vraćanjem dobivene informacije o amplitudi i fazi u zapis pomoću kompleksnog broja, te primjenom inverzne kratkoročne Fourierove transformacije. Budući da smo koristili preklapajuće kratkoročne Fourierove transformacije, konačne uzorke dobivene inverznom kratkoročnom Fourierovom transformacijom trebamo zbrojiti na odgovarajući način, koristeći metodu *Overlap-add*, standardnu metodu u digitalnoj obradi signala (vidi, na primjer, [5]). Pojednostavljen prikaz Fourierove reprezentacije *Phase Vocoder* algoritma dan je na slici 3.2.

Sada smo spremni za detaljan opis rada algoritma, počevši od faze analize, preko faze procesiranja, pa sve do faze sinteze.

3.3 Faza analize

U ovom poglavlju detaljno ćemo opisati fazu analize *Pitch Shifting* algoritma koji se zasniva na *Phase Vocoder* tehnici. Kao što smo već naveli u poglavlju 3.2, faza analize sastoji se od preklapajućih kratkoročnih Fourierovih transformacija (STFT).



Slika 3.2: Pojednostavljen prikaz Fourierove interpretacije *Phase Vocoder* algoritma

Određivanje stvarne frekvencije

Prije opisa same implementacije, osvrnimo se na nužnost uvođenja preklapanja. Najprije ćemo opisati postupak određivanja stvarne frekvencije pojedine frekvencijske komponente (frekvencijskog spremnika), nakon primjene Fourierove transformacije. U tu svrhu, podsjetimo se da je rezultat Fourierove transformacije informacija o amplitudi i fazi parcijalnih frekvencijskih komponenti, koje odgovaraju frekvencijama koje su uniformno raspoređene po promatranom frekvencijskom spektru. Ako označimo $x = \mathrm{Re}(X[k])$ i $y = \mathrm{Im}(X[k])$, amplitudu računamo po formuli

$$A[k] = \sqrt{x^2 + y^2},$$

dok pomak u fazi određujemo izrazom

$$\phi[k] = \mathrm{atan2}(y, x).$$

Znamo da standardna atan2 funkcija ima povratnu vrijednost u intervalu $[-\pi, \pi]$. Intuitivno, taj postupak promatramo na način kao da parcijalne komponente početnog signala stavljamo u odgovarajuće frekvencijske spremnike. Ako je veličina promatranog uzorka (veličina prozora u slučaju STFT) L, a f_s frekvencija uzorkovanja, frekvencija susjednih frekvencijskih spremnika se razlikuje za $\frac{f_s}{L}$ Hz. Problem nastaje ako se u originalnom signalu pojavljuje parcijalna komponenta čija frekvencija se nalazi između dva frekvencijska spremnika. U tom slučaju se informacija o toj parcijalnoj komponenti raspodijeli na susjedne frekvencijske spremnike (engl. *smearing*). Uzrok tog problema su umjetni pomaci u fazi, koji nastaju prilikom podjele polaznog signala na okvire određene duljine. Naime, početni signal uopće ne mora imati pomak u fazi, no ako okvir ne završava točno na kraju njegovog perioda, drugi okvir će sadržavati isti signal, no s umjetnim pomakom u fazi. Tada diskretna Fourierova transformacija dobiveni uzorak neće moći pogoditi s odgovarajućom frekvencijom, što će rezultirati opisanom pojavom. Zaključujemo da razlika u fazi između dva uzastopna prozora za posljedicu ima odstupanje stvarne frekvencije od frekvencije odgovarajućeg frekvencijskog spremnika. Štoviše, ako znamo pomak u fazi, vrlo jednostavno možemo izračunati stvarnu frekvenciju kojoj pripada vrijednost zabilježena određenim frekvencijskim spremnikom, budući da je kutna frekvencija, zapravo, razlika u fazi podijeljena s duljinom vremenskog intervala.

Ako s $\phi[k]_i$ označimo pomak u fazi k-te komponente diskretne Fourierove transformacije i-tog okvira te s Δt razliku u vremenu između dva promatrana okvira, stvarnu frekvenciju k-te komponente možemo računati na sljedeći način:

1. izračunamo devijaciju novodobivene kutne frekvencije od kutne frekvencije k-te komponente, po formuli
$$\Delta\omega[k] = \frac{\phi[k]_i - \phi[k]_{i-1}}{\Delta t} - \omega_{bin}[k] \tag{3.1}$$
2. normaliziramo dobivenu devijaciju na interval $[-\pi, \pi]$
3. dobivenom broju natrag pribrojimo $\omega_{bin}[k]$, te ga podijelimo s 2π, da bismo dobili stvarnu frekvenciju.

Uočimo da je $\Delta t = \frac{o}{f_s}$, gdje je o veličina okvira, a f_s frekvencija uzorkovanja.

Nadalje, uočimo da pomak u fazi k-te frekvencijske komponente računamo koristeći funkciju atan2, koja vraća vrijednosti u intervalu $[-\pi, \pi]$. Zaključujemo da je sve pomake u fazi, koji su inicijalno izvan tog intervala, nemoguće jednoznačno odrediti. Iz tog razloga, potrebno je gušće fragmentirati početni signal. Budući da, zbog rezolucije diskretne Fourierove transformacije, veličinu promatranog okvira ne smijemo previše smanjiti, gustoću fragmentacije postižemo tako da gledamo preklapajuće okvire (za detalje vidi [3]). U konkretnoj implementaciji koristimo preklapanje od 75 %, što je standardna vrijednost u praktičnim primjenama (vidi [13]).

Implementacija STFT

U ovom poglavlju opisujemo kako efikasno računati STFT, koristeći FFT algoritam. Neka je L veličina prozora kojeg promatramo, a w Hannov prozor veličine L (vidi poglavlje 2.3). Također, pretpostavimo da koristimo preklapanje od 75 %. Dakle, razmak između dva preklapajuća okvira je $\frac{L}{4}$ uzoraka. Označimo tu vrijednost s o. Nadalje, označimo ukupni pomak od prvog promatranog uzorka u signalu s m_j, gdje je j indeks promatranog okvira. Očito je $m_j = j \cdot o$. Neka je $W_N = e^{\frac{2\pi i}{N}}$. STFT za j-ti okvir u signalu, po definiciji, računamo po formuli

$$X_j[k] = X(m_j, k) = \sum_{n=-\infty}^{+\infty} x[n]w[n - m_j]W_N^{-kn},$$

za $k = 0, 1, \ldots, N-1$. Uočimo da isti izraz možemo ekvivalentno zapisati na sljedeći način:

$$X_j[k] = \sum_{n=-\infty}^{+\infty} x[n + m_j]w[n]W_N^{-kn},$$

za $k = 0, 1, \ldots, N-1$. Kako promatramo okvir duljine L, beskonačna suma se svodi samo na sumiranje od 0 do $L-1$:

$$X_j[k] = \sum_{n=0}^{L-1} x[n + m_j]w[n]W_N^{-kn}, \tag{3.2}$$

za $k = 0, 1, \ldots, N-1$. Uočimo da je izraz (3.2) upravo diskretna Fourierova transformacija niza

$$\tilde{x}[n] = x[n + m_j]w[n],$$

stoga je možemo efikasno izračunati FFT algoritmom.

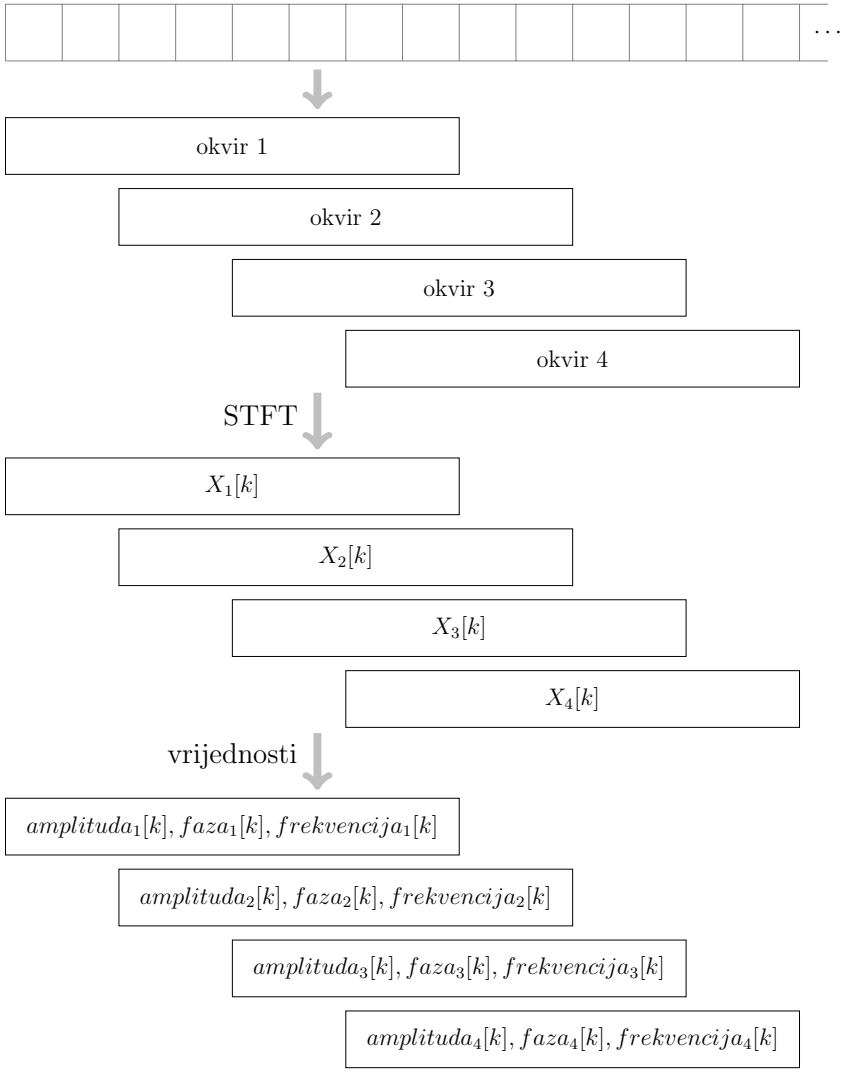
Napomenimo da nema potrebe svaki put računati vrijednosti $w[n]$. Budući da je veličina prozora fiksna te da ju znamo prije izvršavanja programa, razumno je te vrijednosti izračunati jednom, prilikom inicijalizacije, te ih čuvati u memoriji.

Zaključno, faza analize promatranog *Pitch Shifting* algoritma sastoji se od dva koraka:

1. za svaki od okvira potrebno je pomoću FFT algoritma izračunati pripadni STFT
2. za svaku od frekvencijskih komponenti potrebno je izračunati amplitudu i stvarnu frekvenciju.

Na slici 3.3 su grafički prikazani osnovni koraci faze analize za 4 preklapajuća okvira.

U poglavlju 2.2 naglasili smo da se, koristeći iterativni FFT, prvi korak može izračunati u složenosti $\mathcal{O}(n \log n)$, gdje je n veličina okvira (L u našem slučaju).



Slika 3.3: Osnovni koraci faze analize za 4 preklapajuća okvira

Također, važno je još jednom naglasiti da, za korištenje FFT algoritma, L mora biti potencija broja 2. Drugi korak je, očito, linearan u duljini prozora, budući da se za svaku od frekvencijskih komponenti mora napraviti modifikacija opisana izrazom (3.1), uz odgovarajuće mapiranje dobivene vrijednosti u segment $[-\pi, \pi]$. Dakle, zaključujemo da je vremenska složenost faze analize, za svaki od okvira koji procesiramo, jednaka $\mathcal{O}(n \log n)$, u duljini prozora.

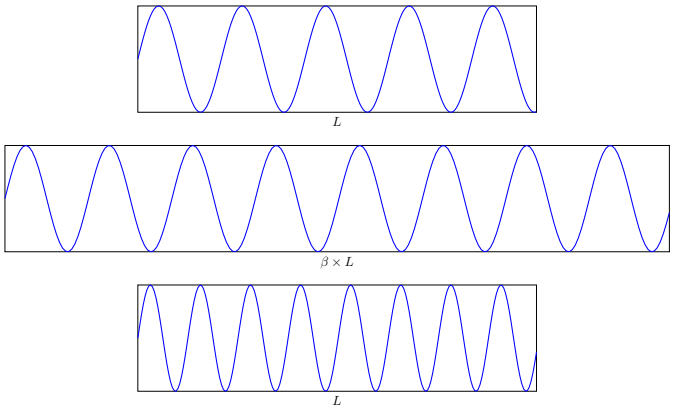
3.4 *Pitch Shifting* ili promjena visine tona

U ovom poglavlju objasnit ćemo na koji se način realizira sam *Pitch Shifting* ili promjena visine rezonirajućeg tona, to jest, bit će opisana faza procesiranja u našoj varijanti *Phase Vocoder* algoritma.

Vratimo se za trenutak u vremensku perspektivu i pretpostavimo da imamo inicijalni signal promatran u nekom vremenskom intervalu duljine L. Uočimo da je, iz te perspektive, za promjenu visine tona za neki faktor skaliranja β, intuitivno potreban sljedeći niz koraka (za detalje vidi [13]):

1. prvo promijenimo duljinu promatranog signala, bez promjene visine tona (bez da utječemo na frekvenciju), na βL,
2. a nakon toga, tako modificirani signal trebamo reproducirati β puta brže, tj. potrebno se vratiti na inicijalnu duljinu vremenskog intervala, no s modificiranom frekvencijom.

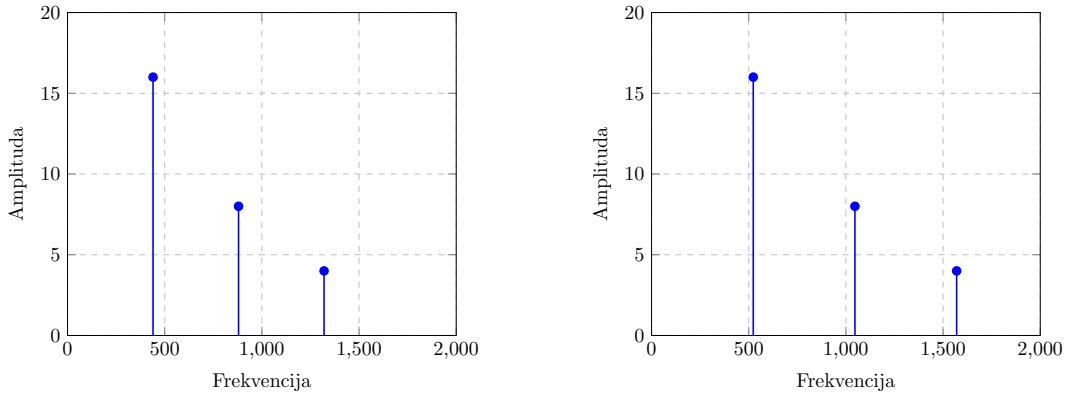
Grafički prikaz opisane modifikacije prikazan je na slici 3.4.



Slika 3.4: Promjena visine tona iz vremenske perspektive

Ako se vratimo u frekvencijsku perspektivu, promjena visine tona je relativno jednostavna. Promotrimo još jednom rezultat prethodne faze, nakon računanja stvarnih frekvencija parcijalnih frekvencijskih komponenti. Za svaki od frekvencijskih spremnika, dobivenih kratkoročnom Fourierovom transformacijom, imamo pripadajuću amplitudu te njegovu stvarnu frekvenciju. Uočimo da se spremljene vrijednosti amplituda ne smiju mijenjati. Naime, već smo više puta naveli da sva svojstva početnog signala, osim pripadajuće frekvencije, ne želimo mijenjati. Ako inicijalni signal frekvencije f_1 ima amplitudu A, očekujemo da i novi modificirani signal frekvncije f_2 ima identičnu amplitudu. Označimo ponovno pripadajući faktor skaliranja s β. Uočimo da je, za promjenu visine tona, dovoljno "proširiti" polje amplituda za faktor β.

Konkretno, iz perspektive Fourierove transformacije, želimo svaku od vrijednosti amplituda povezati s drugim frekvencijskim spremnikom, čija je frekvencija dobivena izrazom βf_{bin}, gdje je s f_{bin} označena frekvencija spremnika uz koji je inicijalno vezana promatrana amplituda. Isti postupak ponovimo i s poljem stvarnih frekvencija. Grafički prikaz opisane modifikacije dan je na slici 3.5.



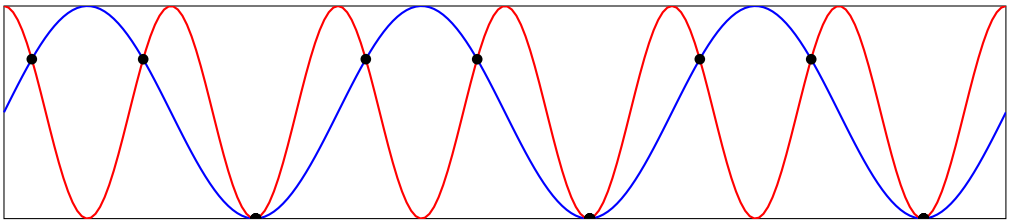
(a) Početne frekvencije

(b) Frekvencije nakon promjene visine tona skaliranjem za neki faktor β

Slika 3.5: Promjena visine tona iz frekvencijske perspektive

Zbog ograničenja broja frekvencijskih spremnika u diskretnoj Fourierovoj transformaciji i njihovih pripadajućih frekvencija, teško je očekivati da ćemo množenjem frekvencije pojedinog spremnika s β "pogoditi" točnu vrijednost nekog drugog spremnika. Taj problem je moguće riješiti na nekoliko načina. Jedna varijanta je primjena linearne interpolacije na način da se vrijednosti amplitude rasporede u dva različita frekvencijska spremnika. Iako primjena linearne interpolacija ima svojih prednosti (vidi [13]), u ovom radu koristit ćemo jednostavniju metodu procjene odgovarajuće frekvencije. Naime, budući da u konkretnoj implementaciji koristimo prozor veličine 2048 (vidi poglavlje 4), uz standardnu frekvenciju uzorkovanja od 44 100 Hz, frekvencije dva susjedna frekvencijska spremnika razlikuju se za približno 21.5 Hz. To je dovoljno mala razlika da se eventualna manja pogreška u određivanju točne frekvencije ne osjeti na modificiranom signalu, u smislu disonantnih tonova. Razlika je tim manja što se, osim informacije o amplitudi, uz novoodređeni frekvencijski spremnik veže i stvarna frekvencija parcijalne komponente, također, skalirana faktorom β. Ideja je za novi frekvencijski spremnik odabrati onaj spremnik čija je frekvencija najbliža po modulu frekvenciji βf_{bin}. Uočimo da na taj način čuvamo točnu vrijednost amplitude i time postižemo da se energija koju signal prenosi ne dijeli na više frekvencijskih spremnika.

Spomenimo da opisanu modifikaciju radimo samo za one frekvencije f za koje vrijedi da je $f < \frac{f_s}{2}$, gdje je f_s frekvencija uzorkovanja, tj. za koje je zadovoljen takozvani Nyquistov kriterij. Naime, Nyquist–Shannonov teorem o uzorkovanju garantira da je, za danu frekvenciju uzorkovanja f_s, moguća potpuna rekonstrukcija polaznog analognog signala ako za najvišu frekvenciju vrijedi da je manja od $f < \frac{f_s}{2}$ (vidi [21]). Na taj način sprječavamo i pojavu koja se u literaturi naziva *aliasing* te označava situaciju kad dani uzorak može odgovarati sinusoidalnim signalima različitih frekvencija (vidi [21]). Primjer te pojave vidimo na slici 3.6. Uočimo da su sve frekvencije tonova koje promatramo (vidi tablicu 1.1) manje od 22 050 Hz, što je polovica standardne frekvencije uzorkovanja od 44 100 Hz, koja je karakteristična za digitalne signale u praksi (vidi [26]).



Slika 3.6: Primjer pojave *aliasing-a*. Dva sinusoidalna signala mogu imati jednake uzorke kad je frekvencija barem jednog od njih veća od polovice frekvencije uzorkovanja.

Određivanje faktora skaliranja

Nakon što smo u prethodnom poglavlju opisali kako mijenjati visinu tona, u ovom poglavlju opisat ćemo na koji način biramo odgovarajući faktor skaliranja. Te teme smo se već dotakli prilikom opisivanja odnosa fundamentalnih frekvencija različitih tonova te opisivanja pojmova harmonije i tonaliteta, u poglavlju 1.3.

Algoritam koji razvijamo, kao ulazne podatke, prima podatak o tonalitetu u kojem želimo generirati harmoniju te interval koji želimo generirati. Moguće vrijednosti intervala su terca, kvarta, kvinta, seksta te oktava. Kao što je navedeno u poglavlju 1.3, kvarta, kvinta i oktava su čisti intervali. Bez obzira na promatrani tonalitet, oni točno propisuju razmak među tonovima (vidi tablicu 1.2) koji je potrebno generirati. Zaključujemo da je u slučaju kvarte, kvinte i oktave, određivanje faktora skaliranja dano izrazom (1.1), gdje sa s označavamo broj čija je apsolutna vrijednost broj polustepena za koji želimo pomaknuti početni ton, dok mu je predznak određen time želimo li početni ton spustiti $(-)$, ili povisiti $(+)$, za dani interval.

U slučaju terce i seksta situacija se znatno komplicira, budući da je potrebno odrediti želimo li generirati mali ili veliki interval. Kao što je objašnjeno u poglav-

lju 1.3, u tome ključnu ulogu igra informacija o tonalitetu. U nastavku ćemo opisati algoritam kojim određujemo broj polutonova s, pomoću kojeg ćemo iz izraza (1.1) dobiti odgovarajući faktor skaliranja u tom (degeneriranom) slučaju.

Za početak, potrebno je odrediti visinu tona koju reprezentira segment signala koji promatramo. Pretpostavljamo da je vremenski interval koji obuhvaća taj segment dovoljno malen, da se u njemu pojavljuje samo jedan ton. Algoritam za određivanje visine tona, kojeg koristimo u ovom radu, bit će posebno opisan u sljedećem odjeljku 3.4. Nakon što odredimo visinu tona, u sljedećem koraku provjeravamo pripada li dobiveni ton danom tonalitetu. Ukoliko ne pripada, tada prema dogovoru (poučeni praktičnim iskustvom), vraćamo broj polustepena koji odgovara velikom intervalu. Ukoliko dobiveni ton pripada danom tonalitetu, potrebno je odrediti relativan pomak dobivenog tona u odnosu na prvi ton tonaliteta, tj. stupanj tog tona u ljestvici tog tonaliteta. U završnom koraku, (cirkularno) prolazimo po ljestvici danog tonaliteta i brojimo polustepene, sve dok se ne pomaknemo za 3 tona (u slučaju terce) ili 6 tonova (u slučaju sekste). Smjer u kojem prolazimo po ljestvici tonaliteta određen je informacijom želimo li početni ton sniziti ili spustiti.

U sljedećem poglavlju pokazat ćemo da prepoznavanje visine tona možemo izuzetno efikasno implementirati. Sve ostale operacije su vrlo jednostavne i svode se na cirkularne prolaze modulo 12, po polju *bool* vrijednosti od 12 elemenata, s adekvatnim pomakom. Točnije, uvodimo polje od 12 *bool* vrijednosti koje će služiti kao maska, prema kojem ćemo određivati pripada li ton određenom tonalitetu ili ne (uočimo da je za to dovoljan jedan *short*, da nema stvarne potrebe za alokacijom polja). Polje je oblika kao što je prikazano na slici 3.7.

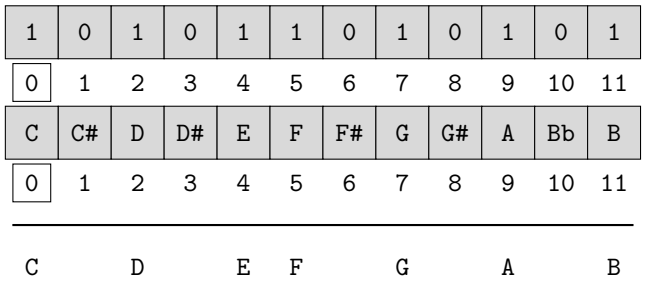
1	0	1	0	1	1	0	1	0	1	0	1
0	1	2	3	4	5	6	7	8	9	10	11

Slika 3.7: Maska za određivanje pripadnosti tonalitetu

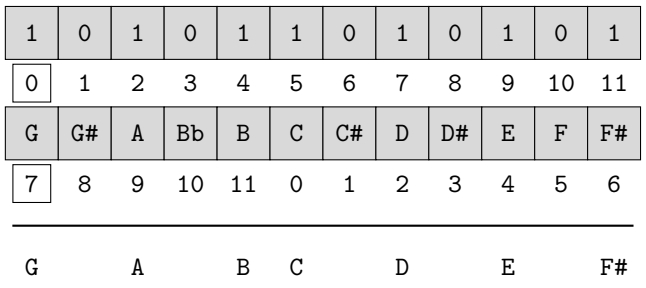
Uočimo da polje tog oblika, zapravo, simulira razmake među tonovima karateristične za dur ljestvicu. Detalji će biti objašnjeni u primjeru 3.4.1.

Primjer 3.4.1. *Pretpostavimo da želimo odrediti sve tonove C dur ljestvice. To vrlo jednostavno možemo učiniti koristeći masku sa slike 3.7. Naime, ukoliko sve oznake tonova poredamo po redu u polje (C, C♯, D, D♯, E, F, F♯, G, G♯, A, B♭, B), dovoljno je samo primijeniti navedenu masku. Vrijednost true u maski označava da ton pripada durskom tonalitetu. U suprotnom, ton ne pripada durskom tonalitetu (vidi sliku 3.8). Ukoliko želimo utvrditi pripadnost G dur ljestvici, potrebno je cirkularno posmaknuti polje oznaka, tako da počinje tonom G (vidi sliku 3.9). Općenito, za*

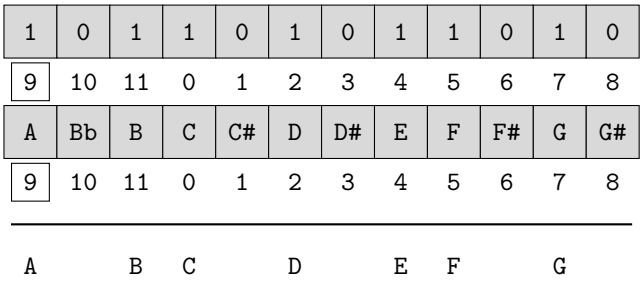
molsku ljestvicu, potrebno je najprije cirkularno posmaknuti masku, da starta od 9. indeksa. Na slici 3.10 prikazan je postupak određivanja pripadnosti A mol tonalitetu. Ovim postupkom vrlo je jednostavno utvrditi i stupanj pojedinog tona u ljestvici.



Slika 3.8: Tonovi C dur ljestvice



Slika 3.9: Tonovi G dur ljestvice



Slika 3.10: Tonovi A mol ljestvice

Prepoznavanje visine tona

Postoje mnogi algoritmi koji se primarno bave prepoznavanjem visine tona za dani glazbeni signal. Mnogi od njih su sofisticirane metode koje su izrazito precizne, no uz njih dolazi određena cijena u vidu potrošnje vremenskih resursa. Pregled nekih od tih metoda dan je u knjizi [12]. Budući da algoritam koji razvijamo treba raditi u realnom vremenu, odlučili smo se za metodu koja, u teoriji, nema stopostotnu preciznost, no izrazito je efikasna u smislu vremenske složenosti te koristi informacije koje su već prikupljene diskretnom Fourierovom transformacijom.

Budući da znamo amplitude vezane uz sve frekvencije iz promatranog spektra, nakon primjene diskretne Fourierove transformacije, te pripadajuće stvarne frekvencije vezane uz svaki pojedini frekvencijski spremnik, za početak određujemo stvarnu frekvenciju uz koju je vezana najveća amplituda. Ta frekvencija dominira nad ostalim frekvencijama te će ona imati ulogu frekvencije koja reprezentira ton koji rezonira. Označimo je s f. Kako smo pretpostavili da, u svakom promatranom trenutku, početni signal reprezentira točno jedan ton, imamo dvije mogućnosti:

- f je približno jednaka fundamentalnoj frekvenciji nekog tona
- f je približno jednaka nekom od harmonika nekog tona.

Promotrimo najprije slučaj kad je f blizu fundamentalne frekvencije nekog tona. To je slučaj koji je od primarne važnosti, tj. slučaj koji se uglavnom i pojavljuje u praksi. Imamo nekoliko načina kojima možemo odrediti ton kojeg predstavlja frekvencija f, a svaki od njih se bazira na nalaženju najbliže vrijednosti iz tablice 1.1 i čitanju odgovarajuće oznake.

Prirodno rješenje, koje se nameće samo po sebi, je čuvanje cijele sortirane tablice u memoriji i pronalaženje najbliže frekvencije binarnim pretraživanjem. Algoritam staje kad frekvenciju f ograničimo s fundamentalnim frekvencijama dva susjedna tona u tablici. Odabiremo indeks onog tona čija je fundamentalna frekvencija po modulu bliža f. Opisani algoritam radi u logaritamskom vremenu, s obzirom na broj elemenata u tablici fundamentalnih frekvencija.

Drugo rješenje koje navodimo ne zahtijeva čuvanje cijele tablice fundamentalnih frekvencija, nego samo njezinu najnižu oktavu. Ideja je prvo odrediti oktavu kojoj pripada dana frekvencija, te tada, u jednom prolazu po (sortiranim) frekvencijama iz te oktave, odrediti najbližu. Ovdje koristimo svojstvo da se frekvencije istih tonova u uzastopnim oktavama razlikuju za faktor dva. Uzmemo najnižu frekvenciju iz tablice fundamentalnih frekvencija, ton C1, te ga množimo brojem 2, sve dok rezultat ne preskoči frekvenciju f. Faktor kojim množimo početnu frekvenciju čuvamo u cjelobrojnoj varijabli k. Množenje s dva je, u tom slučaju, posmak (engl. *shift*) ulijevo. Zatim frekvenciju f podijelimo s k i prođemo po polju u kojem je spremljena prva oktava, u kojem najbliža vrijednost predstavlja traženu visinu tona. Opisani

algoritam je, očito, linearan u veličini tablice, no s relativno malom konstantom. Naime, uočimo da je ukupan broj uspoređivanja maksimalno jednak broju oktava, plus 12 usporedbi za odabir odgovarajućeg tona u oktavi. Uz to, broj množenja je maksimalno jednak broju oktava, uz još jedno dijeljenje.

Oba pristupa su izrazito efikasna s obzirom na fiksni, relativno mali broj elemenata tablice fundamentalnih frekvencija. Razlike u smislu njihovog vremenskog izvršavanja nisu statistički značajne. Za konkretnu implementaciju odabrali smo pristup koji ne zahtijeva spremanje cijele tabliice, dakle metodu koja ne uključuje binarno pretraživanje. Bitno je napomenuti da složenost opisanog algoritma ne ovisi o veličini početnog prozora.

Preostaje još komentirati slučaj kad je frekvencija najviše amplitude, zapravo, jedan od harmonika neke fundamentalne frekvencije. Uočimo da za algoritam, predstavljen u poglavlju 3.4, nije potrebna točna oktava, nego samo stupanj koji rezonirajući ton ima u odabranom tonalitetu. U tom smislu, informacija o točnoj oktavi nije niti potrebna, budući da isti tonovi u različitim oktavama imaju istu ulogu u ljestvici. Zaključujemo da opisani algoritam radi i za slučaj kada je harmonik najviše amplitude, ujedno, jednak istom tonu u nekoj drugoj oktavi (to je slučaj kad je frekvencija harmonika jednaka kf_0, gdje je k potencija broja 2, a f_0 fundamentalna frekvencija).

U ostalim slučajevima, koji su rijeđi u praksi, algoritam se ponaša nepredvidivo, budući da izračunati pomak za određeni interval, može, ali ne mora, dati u potpunosti točan rezultat. No, u implementiranom programskom rješenju pokazalo se da prijašnja napomena ne igra veliku ulogu.

Teoretski se taj problem može riješiti tako da, umjesto frekvencije najviše amplitude, tražimo najnižu netrivijalnu frekvenciju, budući da je fundamentalna frekvencija jednaka najnižoj frekvenciji parcijalne frekvencijske komponente signala. No, da bismo odredili takvu frekvenciju potreban je još jedan, dodatni, korak filtriranja, kojim bismo eliminirali sve frekvencije koje smatramo da su rezultat šuma u signalu. Budući da je inicijalni algoritam bez filtriranja davao zadovoljavajuće rezultate, u smislu dobivenog izlaznog signala, te zbog zahtjeva da algoritam treba raditi u realnom vremenu, taj dodatni korak nećemo uvoditi.

Za kraj, navedimo da je složenost cijelog postupka određivanja visine tona linearna u veličini prozora, budući da za određivanje maksimalne amplitude moramo jednom proći po cijelom polju amplituda, dok je dio koji se tiče procesiranja odabrane frekvencije ograničen konstantom koja ovisi o veličini tablice fundamentalnih frekvencija.

3.5 Faza sinteze

U ovom poglavlju ukratko ćemo opisati fazu sinteze našeg *Pitch Shifting* algoritma. Kao što je to opisano u poglavlju u Fourierovoj interpretaciji *Phase Vocoder* algoritma, korak sinteze se sastoji od inverznih kratkoročnih Fourierovih transformacija, koje se akumuliraju u signal koristeći standardnu *Overlap-add* metodu (vidi poglavlje 2.3).

Kao rezultat faze procesiranja imamo procesirano polje amplituda te procesirano polje stvarnih frekvencija, koje predstavljaju osnovnu informaciju o novom, modificiranom signalu. Za početak, potrebno je odrediti novi pomak u fazi, budući da se frekvencije modificiranog signala razlikuju u odnosu na frekvencije početnog signala. Nakon toga moramo osigurati adekvatan prijelaz između dva okvira. Koristeći slične argumente kao i u poglavlju 3.3, zaključujemo da je dovoljno, fazi koja je vezana uz prethodni okvir, pridodati vrijednost koja je izračunata po formuli $\Delta t * \omega_{real}[k]$, gdje $\omega_{real}[k]$ predstavlja stvarnu frekvenciju vezanu uz promatranu k-tu parcijalnu frekvencijsku komponentu, a Δt vrijeme koje promatramo između dva okvira. Napomenimo da okvire, iako se međusobno preklapaju, procesiramo sekvencijalno, tako da uvijek znamo stvarni pomak u fazi prethodnog okvira. Sada kad znamo informaciju o fazi i amplitudi svake od parcijalnih frekvencijskih komponenti novogeneriranog signala, ponovno prelazimo na kartezijeve koordinate da bismo za svaki frekvencijski spremnik dobili odgovarajući kompleksan broj. Ako je $A_j[k]$ nova amplituda, a $\phi_j[k]$ nova faza k-te frekvencijske komponente u j-tom prozoru, novi $X_j[k]$ računamo po formuli

$$X_j[k] = A_j[k]\cos(\phi_j[k]) + iA_j[k]\sin(\phi_j[k]),$$

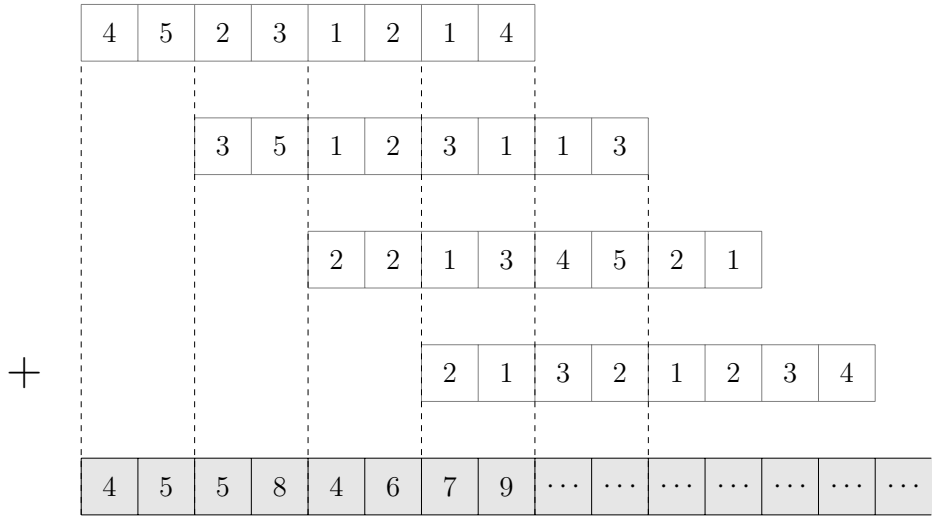
za svaki $k \in \{0, \ldots, N-1\}$.

Sada smo spremni za inverznu STFT niza $X_j[k]$, koju, ponovno, računamo FFT algoritmom, budući da, u suštini, ponovno tražimo diskretnu Fourierovu transformaciju:

$$\tilde{x}_j[n] = \frac{1}{N}\sum_{k=0}^{N-1} X_j[k]W_N^{kn}.$$

U skladu s *Overlap-add* metodom (vidi poglavlje 2.3 i [16]), da bismo dobili konačan signal, potrebno je sve $\tilde{x}_j[n]$ zbrojiti s odgovarajućim pomakom $m_j = j \cdot o$ (vidi sliku 3.11). Budući da smo koristili preklapanje od 75 %, jedan okvir inicijalnog signala duljine L je u potpunosti procesiran i spreman za reproduciranje nakon obrade 4 preklapajuća okvira duljine L.

Želimo postići efekt harmonizacije, tj. želimo da na kraju obrade rezultat bude signal koji reprezentira oba tona istovremeno, i modificirani i početni. Taj efekt postižemo zbrajanjem dva sinusoidalna signala. Iz tog razloga, u zadnjem koraku,



Slika 3.11: *Overlap-add* metoda u fazi sinteze

prije slanja rezultata na audio izlaz, pribrojimo originalni signal modificiranom signalu.

Uočimo da je vremenska složenost faze sinzteze $\mathcal{O}(n \log n)$ u duljini prozora, budući da je za računanje inverzne STFT potreban FFT algoritam. Operacije koje uključuju računanje adekvatnog pomaka u fazi, te akumulacijski korak *Overlap-add* algoritma su linearne u veličini prozora.

Time smo završili opis proučavanog *Pitch Shifting* algoritma.

Poglavlje 4

Implementacija programskog rješenja

U ovom poglavlju predstavit ćemo programsko rješenje koji nudi mogućnost harmonizacije ulaznog zvučnog signala. Ulazni zvučni signal se čita iz glazbene datoteke (ekstenzije *.wav*), harmonizira se u realnom vremenu te se dobivena harmonija reproducira direktno na zadani audio uređaj za reproduciranje, na računalu na kojem se izvršava. Posebnost ovog programskog rješenja je da korisniku daje potpunu kontrolu prilikom biranja željenih intervala harmonizacije te mogućnost odabira tonaliteta u kojem se harmonizacija treba provoditi. Štoviše, programsko rješenje nudi mogućnost mijenjanja različitih harmonija u trenutku izvođenja glazbene datoteke, pri čemu uvelike pomaže fragmentiranje početnog ulaznog signala, objašnjeno u prethodnom poglavlju. Algoritam kojim se provodi harmonizacija je direktna implementacija *Pitch Shifting* algoritma opisanog u poglavlju 3. Programsko rješenje je, u potpunosti, implementirano u programskom jeziku C++. Za početak, navest ćemo najbitnije detalje vezane uz samu implementaciju programskog rješenja. Nakon tog, predstavit ćemo pripadno grafičko sučelje. Na kraju ćemo komentirati efikasnost implementiranog *Pitch Shifting* algoritma, u smislu modifikacije signala u realnom vremenu, što je i bio glavni cilj ovog rada.

4.1 Implementacija *Pitch Shifting* algoritma

Kao što smo već ranije naveli, najprije ćemo navesti najbitnije detalje vezane uz implementaciju samog *Pitch Shifting* algoritma.

Za početak, napomenimo da je veličina okvira ulaznog signala, kojeg ćemo procesirati u jednoj iteraciji algoritma, jednaka $L = 2048$, dok su sve datoteke s kojima radimo uzorkovane frekvencijom $f_s = 44\,100$ Hz. Uočimo da je 2048 potencija broja 2,

što omogućava korištenje FFT algoritma. Nadalje, prozor te veličine daje dovoljno dobru rezoluciju spektra frekvencija koje otkriva diskretna Fourierova transformacija, budući da se frekvencije susjednih frekvencijskih spremnika razlikuju za $\frac{f_s}{L} \approx 21.5$ Hz. Broj uzoraka koji promatramo je dovoljno malen, da možemo smatrati da je signal zahvaćen odabranim okvirom stacionaran u smislu tona, odnosno, frekvencija koje se pojavljuju. Jedan okvir obrađuje $\frac{L}{f_s} \approx 46$ ms ulaznog signala. Taj broj će igrati ključnu ulogu u određivanju može li naš algoritam raditi u realnom vremenu, što će biti komentirano na kraju ovog poglavlja (vidi poglavlje 4.3). Naime, svo procesiranje treba se obaviti u manje od 46 ms, kako krajnji korisnik ne bi osjetio nepravilnosti u izlaznom signalu.

Procesiranja audio ulaza i izlaza

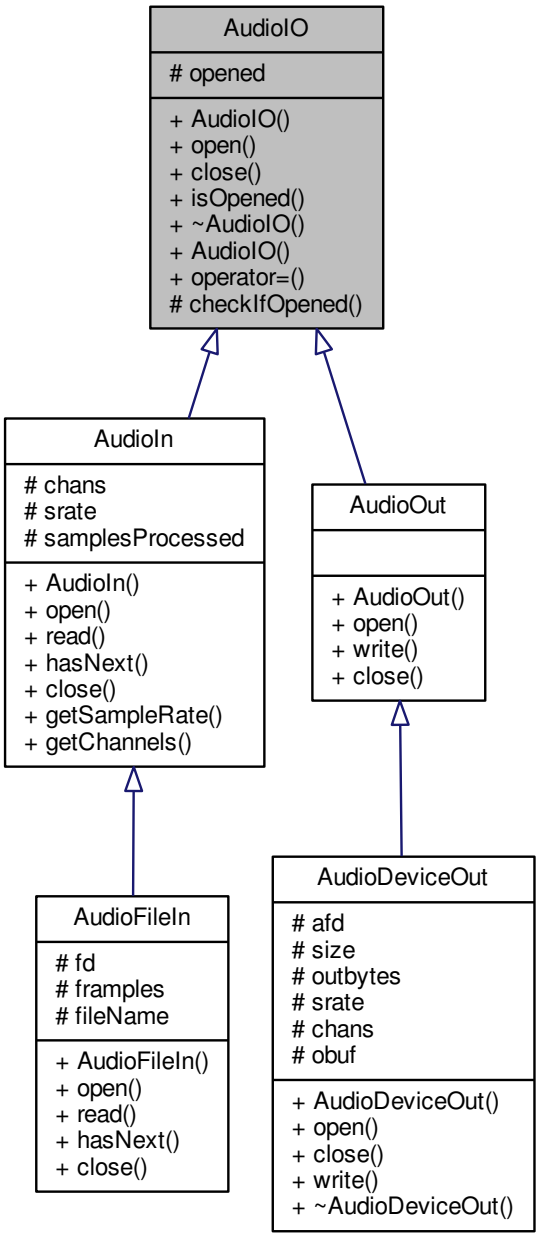
U nastavku ćemo objasniti na koji način je ostvarena komunikacija između programskog rješenja i audio uređaja računala te način na koji je implementirano čitanje audio signala iz glazbene datoteke.

Za komunikaciju s audio uređajima na niskoj razini te za čitanje datoteke ekstenzije *.wav*, koristimo C biblioteku *SndLib*, koja, osim što nudi jednostavno sučelje prema svim audio uređajima na računalu, također, nudi i metode za procesiranje zvučnih datoteka (vidi [24]).

Unutar programskog rješenja ne pozivamo direktno funkcije iz biblioteke, već s bibliotekom komuniciramo preko odgovarajućeg sučelja u duhu objektno orijentiranog dizajna. Za to su zadužene klase koje nasljeđuju apstraktne klase *AudioIn* i *AudioOut*, koje imaju pripadajuće virtualne metode *read*() i *write*() koje omogućuju čitanje, odnosno, pisanje ulaznog signala u datoteku ili na odgovarajući audio uređaj. Klase koje direktno implementiraju tu komunikaciju su, zapravo, omotači oko C sučelja koje nudi *SndLib*. Ovakav dizajn osigurava neovisnost, ne samo o ulaznim i izlaznim audio uređajima koje koristimo, nego i o samoj *SndLib* biblioteci, koja se vrlo lako može zamijeniti bilo kojom drugom bibliotekom koja nudi slično sučelje. Dovoljno je napisati odgovarajuću klasu omotač, koja nasljeđuje neku od opisanih apstraktnih klasa. Dijagram klasa koje su zadužene za audio ulaz, odnosno, izlaz, dan je na slici 4.1.

Implementacija *Pitch Shifting* algoritma

Sam *Pitch Shifting* algoritam implementiran je klasom *PitchShifter*. Svaka instanca klase *PitchShifter* preko konstruktora prima željeni prozor, željenu implementaciju FFT algoritma, te referencu na objekt klase *HarmonyHandler*.

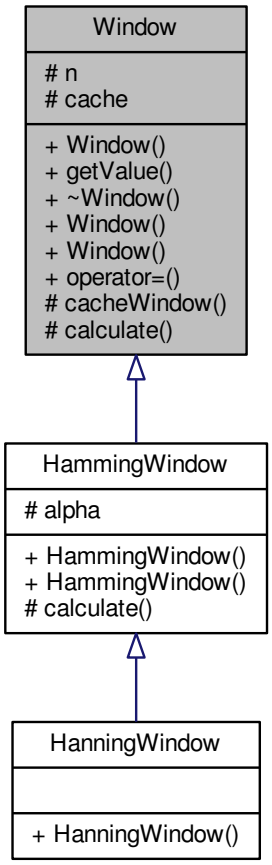


Slika 4.1: UML dijagram klasa zaduženih za upravljanje audio ulazom i izlazom

Klasa *HarmonyHandler* implementira proces harmonizacije, koji se sastoji od određivanja odgovarajućeg faktora skaliranja, na način kako je to opisano u poglavljima 3.4 i 3.4. Konkretno, njezina javna metoda *getPitchShiftFactor*(), na temelju

atributa koji označavaju tonalitet te traženi interval, vraća odgovarajući faktor skaliranja.

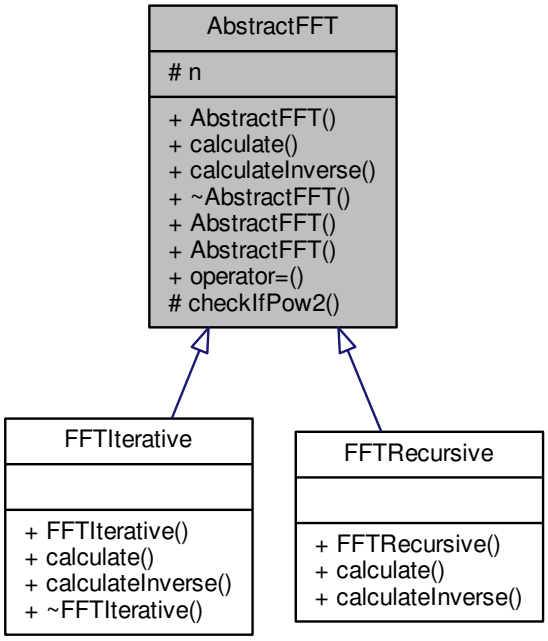
Prozor je svaki objekt koji nasljeđuje apstraktnu klasu *Window* te implementira njezinu apstraktnu metodu *getValue*(). U našem programskom rješenju koristimo Hannov prozor (vidi poglavlje 2.3), koji je implementiran klasom *HanningWindow*. Dijagram klasa, koji prikazuje princip nasljeđivanja klasa koje nasljeđuju Window, prikazan je na slici 4.2. Još jednom napominjemo kako objekti koji nasljeđuju klasu *Window*, prilikom incijalizacije, interno spremaju sve vrijednosti za traženu veličinu prozora, što omogućuje da se tražena vrijednost direktno dohvati iz memorije, bez da se ponovno računa.



Slika 4.2: UML dijagram klasa koje nasljeđuju apstraktnu klassu *Window*

Klasa *AbstractFFT* je apstraktna klasa, koju nasljeđuju klase koje enkapsuliraju implementacije FFT algoritma. Ona ima apstraktne metode *calculate*() i *calculateInverse*(), kojima je uloga na dobivenu memorijsku lokaciju spremiti rezultat dis-

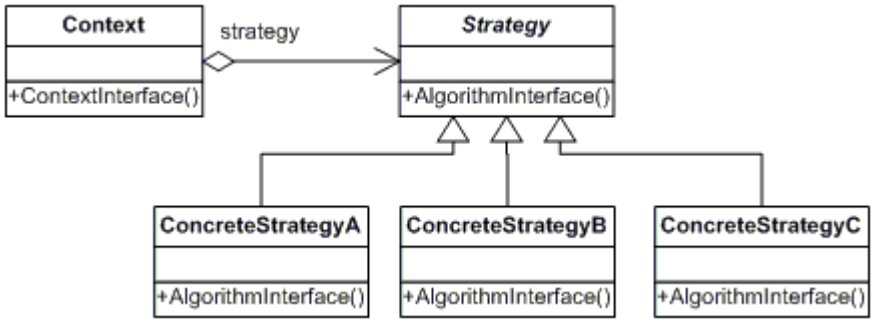
kretne Fourierove transformacije, odnosno, njezinog inverza. U našoj implementaciji koristit ćemo iterativnu implementaciju FFT algoritma, koja se pokazala dovoljno dobrom za naše potrebe, no o tome više u poglavlju 4.3. Slika 4.3 prikazuje dijagram klasa koje nasljeđuju klasu *AbstractFFT*.



Slika 4.3: UML dijagram klasa koje nasljeđuju apstraktnu klassu *AbstractFFT*

Pozivom metode *run*(), objekta klase *PitchShifter*, pokrećemo *Pitch Shifting* algoritam predstavljen u prethodnom poglavlju (vidi poglavlje 3). On se sastoji od svih faza opisanih u tom poglavlju. Njegova implementacija je potpuno neovisna o implementaciji FFT algoritma, prozora koji koristimo, te načina na koji se bira faktor skaliranja. Uočimo da smo klasom *PitchShiter*, zapravo, implementirali oblikovni obrazac strategiju (engl. *Strategy*, vidi [9]). Osnovno svojstvo oblikovnog obrasca strategija je da se algoritam, kojim se rješava neki problem, odabire u *runtime-u*. Osnovna ideja je definirati familiju algoritama (strategija) koji su pogodni za rješavanje nekog problema, svaki od tih algoritama enkapsulirati u vlastitu klasu, te omogućiti da je ostatak implementacije neovisan o implementaciji samog algoritma. Na slici 4.4 prikazan je UML dijagram klasa koji odgovara oblikovnom obrascu strategije. Lako se vidi da taj dijagram direktno možemo primijeniti na naš slučaj, gdje definiramo više različitih strategija, i za prozorsku funkciju i za FFT algoritam.

Spomenimo još da konkretna implementacija *PitchShifter* klase omogućuje koris-



Slika 4.4: UML dijagram klasa oblikovnog obrasca strategije

niku generiranje proizvoljnog broja harmonija. Broj harmonija definiran je konstantom u klasi *HarmonyHandler*, koja je, u našem slučaju, postavljena na 3. Uočimo da je proširenje algoritma iz poglavlja 3, koji bi podržavao proizvoljan broj harmonija, vrlo jednostavno. Naime, jedina promjena je da se od objekta klase *HarmonyHandler* zatraži više faktora skaliranja, ovisno o broju željenih harmonija. Nakon toga se početni signal modificira svakim od tih faktora i svaka od modifikacija se spremi u memoriju. Inverzna diskretna Fourierova transformacije ne mora se računati više od jednog puta. Nakon modifikacije stvarne frekvencije u fazi sinteze, svi dobiveni nizovi se zbroje te se tek onda računa Fourierova transformacija. To omogućuje linearnost diskretne Fourierove transformacije, odnosno, njezinog inverza (vidi poglavlje 2.1). To je bitno, jer je računanje DFT najskuplja operacija u cijelom algoritmu, u smislu vremenske složenosti.

Za kraj, navedimo da se preklapanje postiže na način da je u internom spremniku uvijek $2L$ uzoraka pročitanih iz audio ulaza, gdje je L veličina prozora (2048 u našem slučaju). Tada se čita L elemenata iz internog spremnika, na način da se oni 75 % preklapaju. Nakon 4 iteracije, u potpunosti je procesirano točno L uzoraka te se oni šalju na audio izlaz (uočimo da smo za to morali koristiti $L + \frac{L}{4}$ uzoraka iz internog spremnika). Zatim se druga polovica internog spremnika pomakne za L mjesta ulijevo te se u ostatak internog spremnika učita novih L uzoraka. Na isti način funkcionira i spremnik koji služi kao akumulator prilikom primjene *Overlap-add* metode (vidi poglavlje 3.5). Spomenimo još da je uzorak tipa *double*, tj. da se svo računanje provodi u *double* aritmetici.

4.2 Grafičko sučelje

U ovom poglavlju opisat ćemo osnovne funkcionalnosti i grafičko sučelje programskog rješenja.

Programsko rješenje nudi korisniku sljedeće mogućnosti:

1. odabir bilo koje datoteke ekstenzije *.wav*, u koju je spremljen zvučni zapis kojeg želi modificirati
2. reproduciranje sadržaja te datoteke na zadani audio izlaz računala
3. odabir željenih intervala kojima se postiže efekt harmonizacije te traženog tonaliteta, preko kontrola na grafičkom sučelju. Sve promjene se automatski odražavaju na zvuk koji se trenutno reproducira.

Cjelokupno grafičko sučelje sastoji se od jednog prozora prikazanog na slici 4.5.

Slika 4.5: Grafičko sučelje programskog rješenja

Korisnik odabire datoteku, koja sadrži zvuk kojeg želi harmonizirati, pritiskom na gumb *Browse*. Tada se otvara dijaloški okvir u kojem se odabire željena datoteka. Pritiskom na tipku s ikonom *play*, pokreće se reprodukcija odabranog zvučnog zapisa u novoj dretvi, tako da se ne blokira rad grafičkog sučelja. Promjenom pozicije jedne od četiri rotacijske kontrole s oznakama *Harmony 1, 2, 3*, te *Key*, mijenjaju se odgovarajuće varijable objekta *HarmonyHandler*, te se u sljedećem okviru koji se obrađuje primjenjuje novi interval, odnosno, tonalitet. Brojevi 3, 4, 5, 6 i 8 na *Harmony* rotacijskim kontrolama, redom, označavaju intervale tercu, kvartu, kvintu, sekstu i oktavu.

4.3 Procesiranje u realnom vremenu

Kao što smo već najavili na početku poglavlja, za kraj ćemo, ukratko, još jednom komentirati pojam realnog vremena koji se provlači kroz cijeli rad. Već smo nekoliko puta naveli da pod pojmom procesiranja u realnom vremenu smatramo da se cijela faza procesiranja trenutnog okvira mora obaviti prije nego što završi reproduciranje prethodnog. U našem slučaju, s veličinom prozora $L = 2048$ i frekvencijom uzorkovanja $f_s = 44\,100$ Hz, to znači da za procesiranje imamo približno 46 ms.

Uočimo da ne možemo sa sigurnošću tvrditi da će procesiranje, za svaku arhitekturu, biti u realnom vremenu. Ono što možemo komentirati je vremenska složenost *Pitch Shifting* algoritma, koja je, kad se promotre sve faze, $\mathcal{O}(n \log n)$ u veličini prozora.

Pobrojimo još jednom sve korake algoritma. Nakon što jedan okvir pošaljemo na audio izlaz, najprije moramo pomaknuti ulazni spremnik za L elemenata ulijevo te učitati novi dio uzorka duljine L. Taj dio je, očito, linearan u L. Budući da promatramo 75 % preklapajuće okvire, svaka od faza analize, procesiranja i sinteze mora se ponoviti 4 puta, prije nego što pošaljemo novi okvir na audio izlaz.

Krenimo od faze analize. Prvo je potrebno na trenutni okvir djelovati Hannovim prozorom, što je linearno u duljini prozora. Nakon toga, FFT algoritmom računamo diskretnu Fourierovu transformaciju modificiranog okvira. Taj dio ima vremensku složenost $\mathcal{O}(n \log n)$. Sljedeći korak je, iz dobivene transformacije, izračunati informaciju o amplitudi, pomaku u fazi i stvarnoj frekvenciji. Budući da taj postupak moramo ponoviti za svaku od L vrijednosti u polju, u kojem čuvamo rezultat transformacije, i taj dio je linearan u duljini prozora. Zatim izračunavamo odgovarajući faktor skaliranja. Za intervale kvartu, kvintu i oktavu, faktor skaliranja se računa u konstantnom vremenu, no za tercu i sekstu je potrebno linearno vrijeme, budući da moramo pronaći frekvenciju najveće amplitude. U stvarnoj implementaciji, maksimalnu amplitudu određujemo već prilikom izračunavanja samih amplituda pa je sam dio računanja faktora skaliranja konstantne vremenske složenosti. U najgorem slučaju, on je linearan u veličini tablice fundamentalnih frekvencija, što je neovisno o duljini prozora. Sljedeći korak je stvarno mijenjanje frekvencija, tj. proširivanje polja amplituda i stvarnih frekvencija za izračunati faktor skaliranja. Taj postupak je, također, linearan u veličini okvira. Za kraj, potrebno je izračunati stvarne pomake u fazi (linearno u L), izračunati inverznu transformaciju ($\mathcal{O}(n \log n)$ u veličini okvira L), te pribrojiti dobivene vrijednosti u akumulacijsko polje (linearno u veličini okvira L).

Na testnoj arhitekturi (AMD E1-6015 APU, 1.4 GHz), sve tri faze, ponovljene 4 puta, uspješno završe u vremenu manjem od 46 ms. Kao što je analiza vremenske složenosti pokazala, većina vremena procesiranja ode na 8 brzih Fourierovih tran-

sformacija. Prosječno vrijeme jedne diskretne Fourierove transformacije, iterativnim FFT algoritmom (vidi poglavlje 2.2) kojeg koristimo u ovom radu, na testnoj arhitekturi za 10 000 slučajnih nizova od 2048 *double* vrijednosti, iznosi približno 2 ms. Zbog zadovoljavajućeg vremena izvršavanja, relativno male veličine prozora te činjenice da implementacija grafičkog sučelja, već sama po sebi, uključuje višedretvenost, nismo se odlučili za paralelizaciju FFT algoritma (vidi [4]), kao ni za uvođenje dodatne paralelizacije u sam *Pitch Shifting* algoritam.

Bibliografija

[1] A. Antoniou, *Digital signal processing*, McGraw–Hill, 2016.

[2] B. Benward, *Music in Theory and Practice, Volume 1*, McGraw–Hill Higher Education, 2014.

[3] S. M. Bernsee, *Pitch Shifting Using The Fourier Transform*, `http://blogs.zynaptiq.com/bernsee/pitch-shifting-using-the-ft/`, posjećeno 10. 8. 2017.

[4] T. H. Cormen, Leiserson C. E., Rivest R. L. i Stein C., *Introduction to algorithms*, MIT press, 2009.

[5] R. Crochiere, *A weighted overlap-add method of short-time Fourier analysis/synthesis*, IEEE Transactions on Acoustics, Speech, and Signal Processing **28** (1980), br. 1, 99–102.

[6] M. Dolson, *The phase vocoder: A tutorial*, Computer Music Journal **10** (1986), br. 4, 14–27.

[7] J. L. Flanagan i R. M. Golden, *Phase vocoder*, Bell Labs Technical Journal **45** (1966), br. 9, 1493–1509.

[8] E. Fleischman, *WAVE and AVI Codec Registries*, `https://tools.ietf.org/html/rfc2361`, posjećeno 5. 9. 2017.

[9] E. Gamma, *Design patterns: elements of reusable object-oriented software*, Pearson Education India, 1995.

[10] F. J. Harris, *On the use of windows for harmonic analysis with the discrete Fourier transform*, Proceedings of the IEEE **66** (1978), br. 1, 51–83.

[11] P. Kabal, *Audio File Format Specifications*, `http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/WAVE.html`, posjećeno 5. 9. 2017.

[12] A. Klapuri i M. Davy, *Signal processing methods for music transcription*, Springer Science & Business Media, 2007.

[13] J. Laroche i M. Dolson, *New phase-vocoder techniques for pitch-shifting, harmonizing and other exotic effects*, Applications of Signal Processing to Audio and Acoustics, 1999 IEEE Workshop on, IEEE, 1999, str. 91–94.

[14] N. Macon i A. Spitzbart, *Inverses of Vandermonde matrices*, The American Mathematical Monthly **65** (1958), br. 2, 95–100.

[15] S. K. Mitra i Y. Kuo, *Digital signal processing: a computer-based approach*, sv. 2, McGraw–Hill Higher Education, 2006.

[16] H. S. Nawab i T. F. Quatieri, *Short-Time Fourier Transform*, `https://www.ece.cmu.edu/~ece491/lectures/L29/L0_Chap_STFT.pdf`, posjećeno 11. 8. 2017.

[17] B. Nettles i R. Graf, *The chord scale theory & jazz harmony*, Advance music, 1997.

[18] A. V. Oppenheim, *Discrete-time signal processing*, Pearson Education India, 1999.

[19] M. Plumbley i S. Dixon, *Tutorial: Music Signal Processing*, `https://www.eecs.qmul.ac.uk/~markp/2012/PlumbleyDixon12-ima-tutorial-slides.pdf`, posjećeno 10. 8. 2017.

[20] M. Portnoff, *Implementation of the digital phase vocoder using the fast Fourier transform*, IEEE Transactions on Acoustics, Speech, and Signal Processing **24** (1976), br. 3, 243–248.

[21] J. G. Proakis i D. G. Manolakis, *Digital signal processing 3 rd edition*, Prentice Hall, 1996.

[22] T. F. Quatieri, *Discrete-time speech signal processing: principles and practice*, Pearson Education India, 2006.

[23] D. M. Randel, *The Harvard dictionary of music*, sv. 16, Harvard University Press, 2003.

[24] B. Schottstaedt, *SndLib*, `https://ccrma.stanford.edu/software/snd/snd/sndlib.html`, posjećeno 5. 8. 2017.

[25] B. A. Shenoi, *Introduction to digital signal processing and filter design*, John Wiley & Sons, 2005.

[26] J. Watkinson, *The art of digital audio*, Taylor & Francis, 2001.

Sažetak

U ovom radu proučavamo problem automatske harmonizacije zvučnog signala. Pod pojmom harmonizacije smatramo proces generiranja više različitih signala iz polaznog signala koji, spojeni zajedno, imaju smisla u promatranom glazbenom kontekstu. Harmonizacija se ostvaruje modifikacijom postojećeg signala u frekvencijskoj domeni, koja rezultira promjenom visine tona reprezentiranog početnim signalom. Detaljno je opisan algoritam kojim je proces harmonizacije realiziran. On se bazira na poznatoj tehnici u digitalnoj obradi signala koja se naziva *Phase Vocoder*. Sastoji se od nekoliko faza, u kojima glavnu ulogu ima efikasna implementacija diskretne Fourierove transformacije. Poseban naglasak stavljen je na automatsko određivanje odgovarajućeg pomaka u visini tona u procesu harmonizacije, s obzirom na promatrani glazbeni kontekst, i na efikasnost algoritma u smislu vremenske složenosti. Također, predstavljeno je programsko rješenje koje se temelji na opisanom algoritmu. Glavno svojstvo predstavljenog programskog rješenja je automatska harmonizacija zvučnog signala u realnom vremenu.

Summary

In this paper, we are studying the problem of automatic harmonization of an audio signal. Harmonization is a process of generating different audio signals from one initial signal which, when played together, make sense in a given musical context. Harmonization is realized by modifying the initial signal in a frequency domain. This results in shifting the pitch of the original audio signal. The algorithm presented in this paper is based on the well-known technique in digital signal processing called *Phase Vocoder*. It consists of several phases built around the efficient implementation of discrete Fourier transform. Also, the technique for automatic pitch shift interval selection is presented in the given musical context. The main result of the paper is the application for real time harmonization of a given audio signal based on the presented algorithm.

Životopis

Rođen sam 20. 7. 1993. godine u Čakovcu, gdje sam završio osnovnu i srednju školu (Gimnazija Čakovec, Prirodoslovno–matematički smjer) te osnovnu glazbenu školu. Završio sam Preddiplomski sveučilišni studij Matematika na Matematičkom odsjeku PMF-a, gdje sam 2015. upisao Diplomski sveučilišni studij Računarstvo i matematika na Matematičkom odsjeku PMF-a.

Dobitnik sam nagrade za najuspješnije studente završnih godina diplomskih studija 2017. godine.

Tijekom studija imao sam priliku raditi na nekoliko projekata vezanih uz razna područja matematike i računarstva. Zajedno s kolegicom Anom Paliska, pod vodstvom dr. sc. Tomislava Šmuca i doc. dr. sc. Goranke Nogo, radio sam na projektu *Prepoznavanje glazbenih akorda koristeći tehnike strojnog učenja*, za koji smo akademske godine 2016./2017. nagrađeni Rektorovom nagradom. Za potrebe projekta koristili smo skup podataka koji se sastojao od pjesama J. S. Bacha te *SVM* algoritam, pomoću kojeg smo generirali model za prepoznavanje glazbenih akorda. Od ostalih projekata, ističe se timski projekt pod nazivom *Problem bojanja grafova i Sudoku* kojeg sam radio u suradnji s Anom Paliska i Tomislavom Bujanovićom. Prikazali smo poznatu *Sudoku* zagonetku kao problem bojanja grafova te smo taj problem rješavali pomoću evolucijskih algoritama.

Od kolovoza do listopada 2016. radio sam kao gostujući student na institutu Forschungszentrum Jülich – Jülich Supercomputing Centre u gradu Jülichu u Njemačkoj, jednom od najvećih *HPC* centara u Europi. S mentorom dr. Andreasom Kleefeldom radio sam na implementaciji morfoloških operacija na slikama u boji, baziranih na Loewnerovom uređaju i Einsteinovom zbrajanju za GPU. Rezultat mog rada je C++ (CUDA) biblioteka koja omogućava korištenje morfoloških operacija te interna publikacija u okviru instituta. Također, u okviru instituta, odslušao sam tečajeve *Introduction to parallel programming with MPI and OpenMP* i *GPGPU–Programming using CUDA*, za koje sam primio službene certifikate.

Trenutno radim kao backend developer u kompaniji Trikoder d.o.o. Tijekom studija, obavljao sam još nekoliko studentskih poslova. U suradnji s kompanijama IDE 3 d.o.o. i PJR d.o.o., zajedno s kolegicom Anom Paliska, sudjelovao sam u izradi CRM

softvera. Za tvrtku IDE3 IT d.o.o. izradio sam aplikaciju za razmjenu podataka između dviju baza podataka.

U slobodno vrijeme, aktivno se bavim glazbom. Sviram nekoliko instrumenata te plešem. Tijekom školovanja aktivno sam se bavio rukometom, danas rekreativno igram košarku. Putujem kad god se ukaže prilika.