# Fourierove transformacije u računalnoj grafici

**Mikulan, Sandro**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Sandro Mikulan**

# FOURIEROVE TRANSFORMACIJE U RAČUNALNOJ GRAFICI

**DIPLOMSKI RAD**

**Varaždin, 2023.**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Sandro Mikulan**

**Matični broj: 0016136470**

**Studij: Organizacija poslovnih sustava**

# FOURIEROVE TRANSFORMACIJE U RAČUNALNOJ GRAFICI

**DIPLOMSKI RAD**

**Mentor:**

Doc. dr. sc. Ivan Hip

**Varaždin, rujan 2023.**

*Sandro Mikulan*

**Izjava o izvornosti**

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

*Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi*

---

**Sažetak**

Ovaj rad bavi se Fourierovim transformacijama, postupkom koji je francuski matematičar Jean Baptiste Joseph Fourier razvio početkom 19. stoljeća i koji omogućava dekompoziciju signala na sastavne komponente. Rad daje matematičku pozadinu Fourierovih transformacija kroz objašnjenje Fourierovog reda i samih Fourierovih transformacija. Postavlja se teza da su brze Fourierove transformacije, algoritam koji omogućava drastično brže izračunavanje Fourierovih transformacija na velikim skupovima podataka jedan od najvažnijih algoritama u modernom društvu bez kojeg brojne moderne tehnologije ili ne bi postojale ili bi bile puno kasnije usavršene. Teza se nastoji dokazati kroz pregled praktičnih slučajeva korištenja brzih Fourierovih transformacija posebno u računalnoj grafici. Na kraju se kroz praktični primjer prikazuju implementacije najraširenijih slučajeva korištenja brzih Fourierovih transformacija - za obradu slika, konkretno filtriranje, kompresiju i izoštravanje.

**Ključne riječi:** Fourierove transformacije, diskretne Fourierove transformacije, brze Fourierove transformacije, Fourierov red, računalna grafika, vremenska domena, frekvencijska domena, analiza signala, obrada slika, kompresija, izoštravanje, zamućivanje, filtriranje

.

# Sadržaj

# 1. Uvod

Ovaj diplomski rad bavi se temom Fourierovih transformacija (*engl. Fourier transform*) te primjenom Fourierovih transformacija u računalnoj grafici. Fourierove transformacije na prvi pogled nisu previše atraktivna tema. Razlog tome može se odmah vidjeti kada se pogleda jedna od definicija Fourierove transformacije: „Fourierova transformacija je matematička funkcija koja rastavlja neki signal, koji je vremenska funkcija, u frekvencije od kojih je sačinjen. Rezultat Fourierove transformacije je kompleksna funkcija frekvencija“ [1]. Navedena definicija daje osnovni opis što Fourierova transformacija radi, tj. govori nam da je to funkcija koja kao ulaz prima funkciju i kao izlaz daje funkciju, ali kako bi se razumjelo kako se točno ulazna funkcija pretvara (transformira) u izlaznu potrebni su puno detaljniji opisi i definicije.

U literaturi postoji mnogo definicija Fourierove transformacije ali rijetko koja na jednostavan način objašnjava što je ona i čemu služi. Iz navedenoga se može zaključiti da su Fourierove transformacije kompleksna tema te da je za njihovo razumijevanje potrebna znatna količina predznanja.

Cilj ovog diplomskog rada jest na jednostavan i razumljiv način objasniti što su to Fourierove transformacije, koja je matematička pozadina Fourierovih transformacija te kako se one uklapaju u naš svakodnevni život i u polje informatike, točnije u polje računalne grafike. U suštini cilj je na razumljiv način pokazati zašto su Fourierove transformacije a pogotovo brze Fourierove transformacije *(engl. Fast Fourier Transform, FFT*) jedan od najvažnijih algoritama u povijesti, što dokazuje i izjava Daniela Rockmore-a, poznatog profesora informatike, koji opisuje brze Fourierove transformacije kao algoritam koji „cijela obitelj može koristiti“ i navodi da su brze Fourierove transformacije sveprisutan algoritam koji se danas koristi za analizu i manipulaciju digitalnih ili diskretnih podataka [2].

Rad se sastoji od tri glavna dijela, u prvom dijelu daje se detaljan pregled Fourierovih transformacija, povijesti Fourierovih transformacija i koncepata bitnih za njihovo razumijevanje, u drugom dijelu daje se pregled slučajeva korištenja Fourierovih transformacija s naglaskom na polje računalne grafike. Ovdje treba napomenuti da se rad fokusira na primjene u računalnoj grafici dok su u stvarnosti primjene puno raširenije. Fourierove transformacije imaju svoje primjene u poljima

fizike, medicine, matematike, analize zvuka, inženjerstva, elektronike itd. U trećem dijelu prikazuje se izrada i korištenje aplikacije koja služi za demonstraciju tipičnih primjena Fourierovih transformacija u računalnoj grafici. Navedena aplikacija može se koristiti u nastavi, na kolegijima koji se bave računalnom grafikom i digitalnom obradom slika i dostupna je na poveznici: FFT Demo (xeno317.github.io/Diplomski/).

# 2. Metode i tehnike rada

Pri izradi teorijskog dijela ovog diplomskog rada glavni izvori podataka bile su razne web stranice, znanstveni članci te knjige dostupne online. Za vizualizaciju podataka potrebnih za objašnjavanje Fourierovih transformacija bit će korišten alat SageMath te „Desmos“ grafički kalkulator [3]. SageMath [4] je sustav za računalnu algebru koji pokriva i širok spektar numeričkih metoda koncepata pa tako i Fourierove transformacije. Za vizualizaciju sinusoidalnih rešetki u poglavlju koje se bavi povezanošću Fourierovih transformacija i računalne grafike korišten je Python programski jezik s numpy i matplotlib bibliotekama.

Za potrebe praktičnog dijela rada razvijena je web aplikacija, a za potrebe razvoja korišten je alat Visual Studio Code. Sama arhitektura aplikacije napisana je pomoću HTML-a CSS-a i JavaScript programskog jezika. Za dio koji se odnosi na Fourierove transformacije korištene su dvije javno dostupne biblioteke koje sadrže funkcije potrebne za provođenje Fourierovih transformacija. Konkretno, korištene su biblioteke Mathjs [5] i Project Nayuki [6]. Mathjs je opsežna matematička biblioteka koje uz funkcije za Fourierove transformacije sadrži veliki broj drugih funkcija dok je Project Nayuki biblioteka specijalizirana za Fourierove transformacije.

Sva testiranja i mjerenja brzine izvođenja brzih Fourierovih transformacija u praktičnom dijelu obavljena su na računalu sljedećih specifikacija:

- Procesor: Intel Core i5-7200U 2.5GHz
- Grafička kartica: NVIDIA GeForce GTX 960M (2 GB VRAM)
- RAM – 8 GB DDR4
- Operativni Sustav: Windows 10 pro

# 3. Razrada teme

## 3.1. Povijest Fourierovih transformacija

Kako bi se bolje razumjela važnost Fourierovih transformacija i brzih Fourierovih transformacija u ovom poglavlju će se ukratko proći kroz povijest Fourierovih transformacija, od njihovog otkrića do pojave brzih Fourierovih transformacija.

Fourierove transformacije nazvane su po francuskom matematičaru Jean Baptiste Joseph Fourieru (Auxerre, 21.3.1768 – Pariz, 16.5.1830). Jedno od područja kojima se Fourier bavio bila je teorija topline, točnije Fourier je pokušavao objasniti kako se toplina širi kroz različite materijale tijekom određenog vremenskog perioda. Njegov rad na tom području je rezultirao teorijskom pozadinom jednadžbe provođenja topline (*engl. heat equation*). Za potrebe rješavanja toplinske jednadžbe Fourier je razvio tzv. Fourierovu transformaciju. Fourier je otkrio način kako se složene funkcije mogu rastaviti u jednostavne sinusoide (sinusi i kosinusi) koje zbrojene zajedno čine originalnu funkciju. Ta činjenica mu je pomogla da originalnu jednadžbu za širenje topline pretvori u jednostavniju jednadžbu koja se lakše rješava. Svoje otkriće kao i teorijsku osnovu Fourierove transformacije Fourier je objavio 1822. godine knjizi o analitičkoj teoriji topline (*engl. The Analytical Theory of Heat*) [7].

Nakon toga mnogi znanstvenici i autori proširili su originalni oblik Fourierove transformacije u oblik koji je poznat danas [7], a s vremenom su se pojavili mnogi slučajevi primjene. Sve valne pojave (npr. zvuk) su potencijalni slučajevi korištenja jer Fourierove transformacije omogućuju njihovu analizu.

Fourierove transformacije vjerojatno ne bi bile ni približno važne kao što su danas da nisu otkrivene brze Fourierove transformacije. Jedan od problema Fourierovih transformacija je brzi rast broja potrebnih izračuna s rastom broja podataka s kojima se radi i s povećanjem složenosti funkcije (signala) koja se analizira, jer povećanje složenosti sa sobom povlači veći broj potrebnih podataka potrebnih za kvalitetnu analizu [8].

Otkriće brzih Fourierovih transformacija uvelike dugujemo velikom broju testiranja nuklearnih bombi tijekom Hladnog rata, točnije inicijativi kojom su se htjela zabraniti testiranja. Naime kada je SAD 1954. testirao prvu u seriji svojih litij-deuterijskih termonuklearnih bombi u testu zvanom *„Castle Bravo“* ispostavilo se da je eksplozija

bombe bila puno jača od očekivanja što je rezultiralo puno većim područjem koje je ozračeno. Nakon tog incidenta pojavila se snažna inicijativa da se u potpunosti zabrani testiranje, ali problem je bio kako osigurati da sve strane poštuju zabranu. Inicijativom se željelo zabraniti atmosfersko, podvodno i podzemno testiranje. Za osiguranje poštivanja zabrane atmosferskog i podvodnog testiranja već su postojali pouzdani načini kontrole, ali kontroliranje zabrane podzemnog testiranja je predstavljao problem i tu su se kao rješenje pojavile Fourierove transformacije. Ideja je bila koristiti seizmografe za detektiranje podzemnih testova, ali iz neobrađenih podataka koje seizmograf bilježi nemoguće je razlikovati podzemni test od npr. potresa. Također je bilo nemoguće otkriti dubinu testa i jačinu bombe iako su svi ti podaci bili zabilježeni na seizmografu. Pomoću Fourierovih transformacija originalni podaci bi se rastavili na komponentne frekvencije te bi se iz toga mogle vidjeti frekvencije specifične za detonaciju nuklearne bombe. Jedini problem je bio to što klasične Fourierove transformacije zahtijevaju veliki broj izračuna što je značilo da bi bile potrebne godine da se analiziraju podaci s nekoliko stotina seizmografa koji su pristizali svaki dan [8].

Stoga je bilo važno smanjiti broj potrebnih izračuna na razinu koja bi bila prihvatljiva. Mnogi matematičari i znanstvenici radili su na projektu „optimiziranja“ Fourierovih transformacija te se nakon nekoliko godina došlo do algoritma koji danas nazivamo brze Fourierove transformacije. Matematičar zaslužan za otkriće bio je James Cooley, dok je matematičar John Tukey realizirao prvu računalnu implementaciju brzih Fourierovih transformacija koju je programirao na bazi algoritma koji je otkrio James Cooley. Nažalost ovaj algoritam nije otkriven na vrijeme, od 1963. do 1996. većinom su se neometano provodili podzemni testovi, sve dok 1996. nije donesena zabrana svih vrsta testiranja za koju je u jednoj mjeri ipak zaslužno otkriće brzih Fourierovih transformacija [8].

Unatoč neuspjehu onoga za što su brze Fourierove transformacije originalno bile namijenjene njihovo otkriće otvorilo je put za razvoj fizike, matematike, medicine, informatike, računalstva i mnogih drugih polja. Danas kad god se negdje obrađuje neka vrsta signala vrlo je vjerojatno da je u proces obrade uključen neki oblik brzih Fourierovih transformacija.

## 3.2. Fourierov red

Fourierov red (*engl. Fourier series*) može se smatrati posebnim slučajem Fourierovih transformacija bez čijeg razumijevanja ne možemo u potpunosti razumjeti Fourierove transformacije, stoga će se u ovom poglavlju ukratko objasniti što je Fourierov red i koja je njegova povezanost s Fourierovim transformacijama.

Za početak pogledajmo dvije definicije Fourierovog reda. Prva definicija govori da: „Fourierov red rastavlja periodičku funkciju u sumu sinusoidalnih funkcija“[7]. Dok druga definicija definira Fourierov red kao razvoj periodičke funkcije $f(t)$ u beskonačnu sumu sinusa i kosinusa [9]. Vidi se da obje definicije spominju periodičke funkcije, pa je prvo potrebno definirati što je to periodička funkcija.

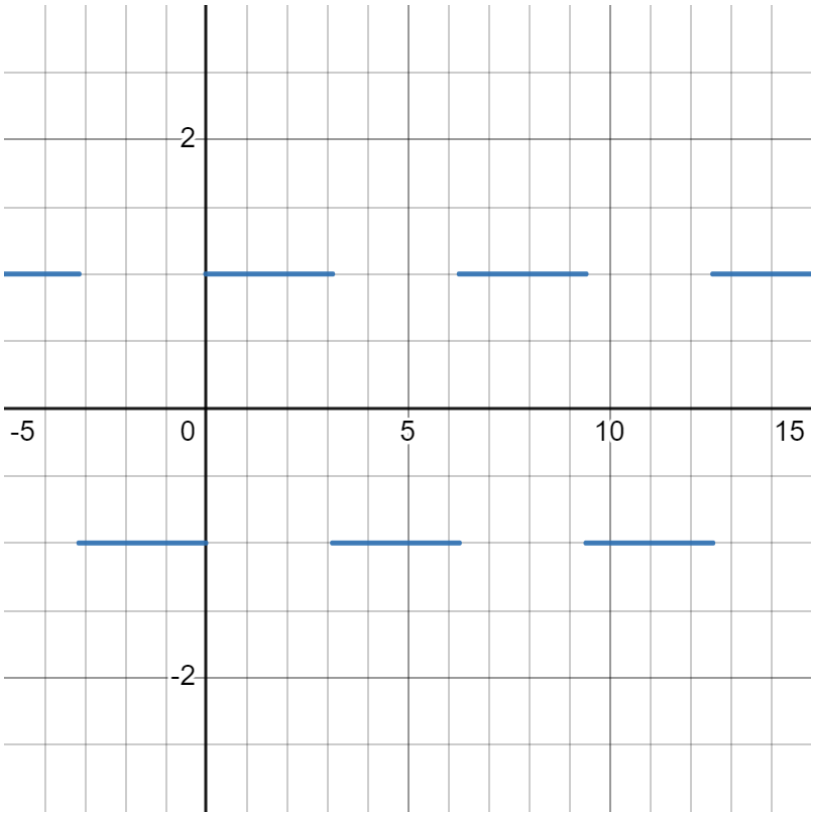
Funkcija $f(t)$ s osnovnim periodom $T$ je periodička ako je sljedeća jednadžba istinita za sve vrijednosti $t$:

$$f(t+T) = f(t)$$

Ako argument funkcije ima dimenziju vremena period se može definirati kao najmanje vremensko razdoblje veće od 0 koje je takvo da će nakon svakog intervala duljine $T$ funkcija imati istu vrijednost. Dakle funkcija je periodička ako vrijedi da funkcija ima istu vrijednost $f(t)$ nakon svakog perioda $T$ u kojem god vremenskom razdoblju promatrali funkciju.

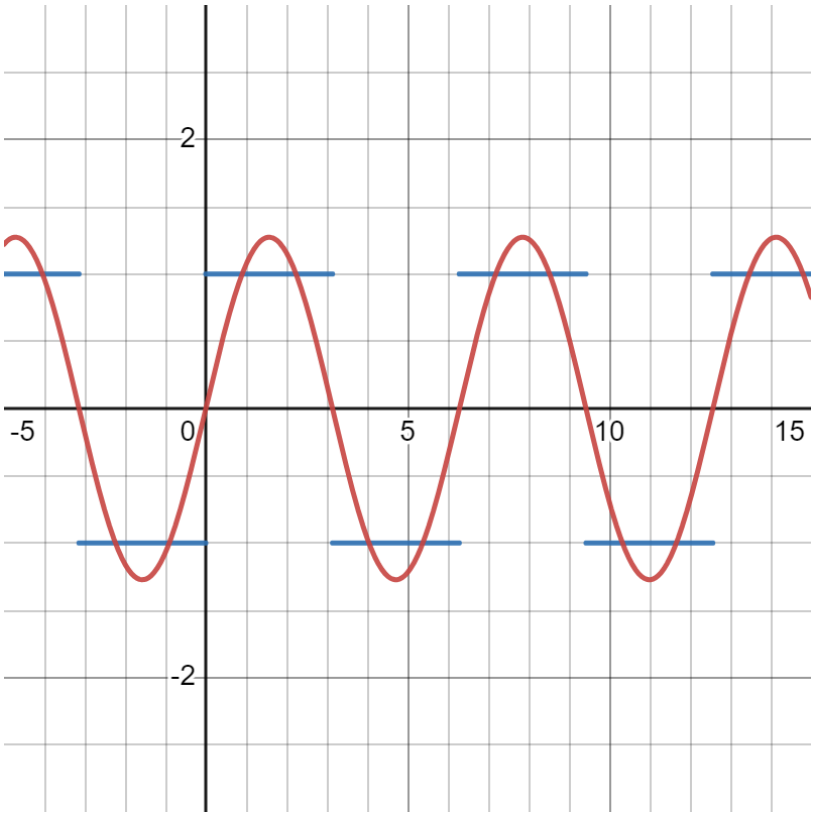
Postavlja se pitanje: ako imamo periodičku funkciju $f(t)$ koliko dobro možemo aproksimirati tu funkciju pomoću sume sinusa i kosinusa? Dirichletov teorem garantira da se svaka funkcija koja je po dijelovima neprekidna i čija je derivacija po dijelovima neprekidna može razviti u Fourierov red (po dijelovima neprekidna ugrubo znači da funkcija ima konačan broj prekida na konačnom intervalu) [10] [11].

Kao konkretan primjer Fourierovog reda za neku periodičku funkciju često se uzima primjer pravokutnog vala (*engl. square wave*) koji je prikazan na slici 1.




*Slika 1: Primjer pravokutnog vala [autorski rad]*

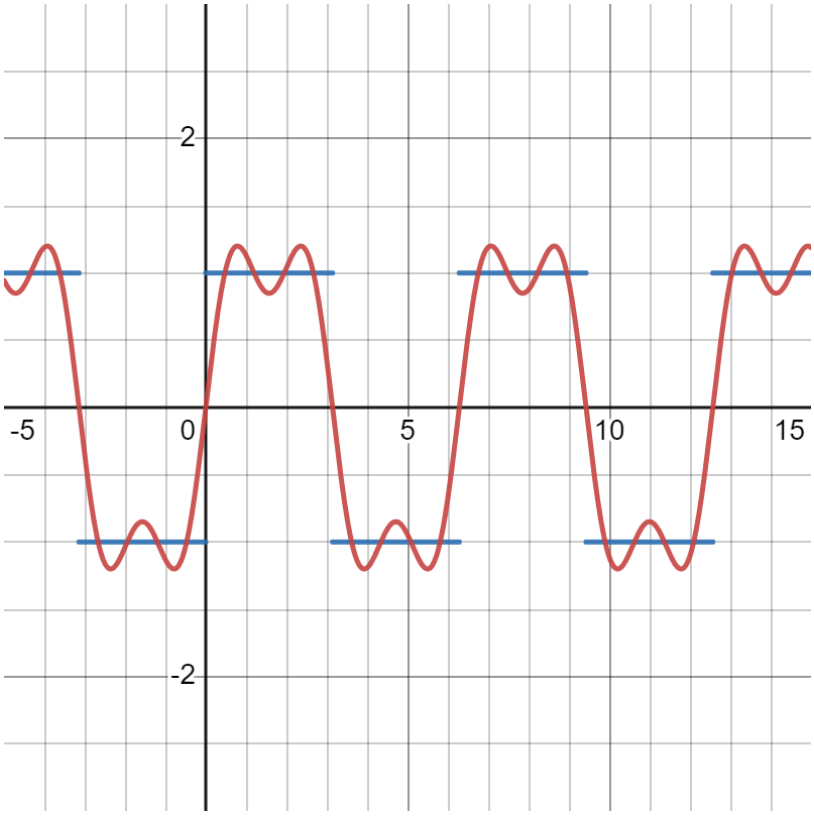
Prikazana funkcija u ovom primjeru ima period $\pi$, te oscilira između vrijednosti 1 i -1. Ono što se želi postići je pronaći određeni broj sinusoida koje dovoljno dobro aproksimiraju gornju funkciju. Ako se uzme funkcija $f_1(t) = \frac{4}{\pi}\sin(t)$ te se ona iscrta uz našu originalnu funkciju dobiva se stanje prikazano na slici 2 [12] [13].




*Slika 2: Originalna funkcija i prvi član Fourierovog reda [autorski rad]*

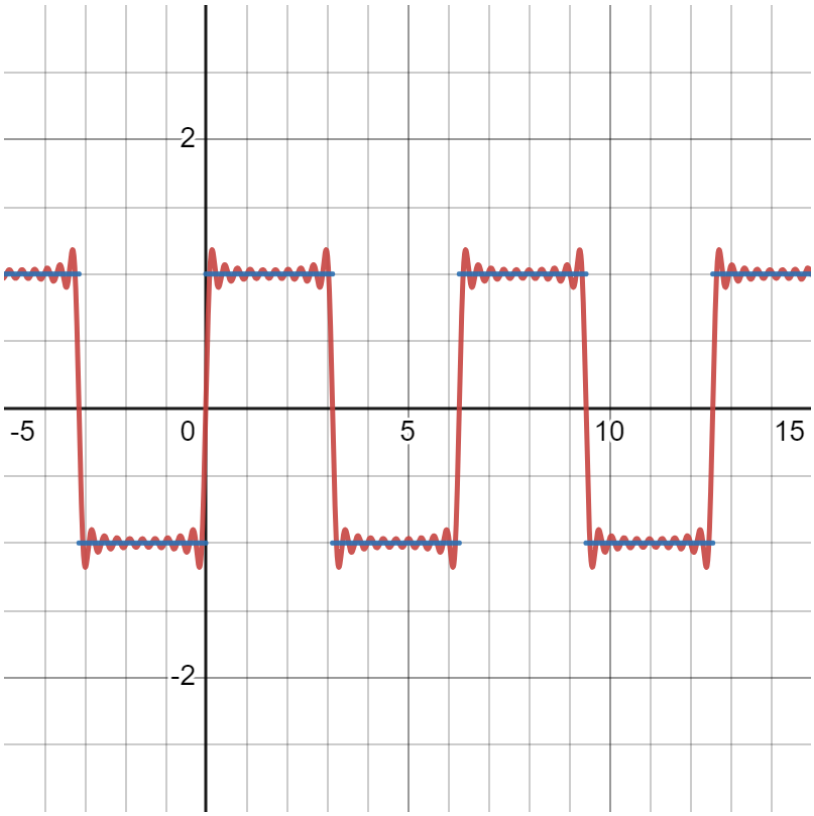
Može se primijetiti da navedena funkcija otprilike prati tijek originalne funkcije ali je dosta loša aproksimacija. Kako bi se dobila bolja aproksimacija potrebno je više članova, dakle još funkcija koje će se zbrojiti s prvom funkcijom. Ono što želimo postići ako želimo bolju aproksimaciju je dopuniti prvi član kako rezultat ne bi prolazio „iznad“ i „ispod“ originalne funkcije. Kada zbrajamo dvije funkcije možemo razmišljati tako da na mjestima gdje obje funkcije rastu/padaju rezultantna funkcija će također rasti/padati tj. povećat će joj se vrijednost, a na mjestima gdje jedna funkcija raste, a druga funkcija pada u rezultantnoj funkciji rast i pad će se poništiti, što će rezultirati manjom vrijednosti. Ako gledamo naš prvi član vidimo da bi se željeni rezultat postigao ako uzmemo funkciju koja ima tri puta manji period od prvog člana. Također je potrebno uzeti funkciju s manjim koeficijentom od prvog člana, kako se u rezultatu funkcije ne bi u potpunosti poništile, stoga uzimamo funkciju: $f_2(t) = \frac{1}{3}\sin(3t)$. Ova funkcija je drugi član parcijalne sume. Ovdje uvodimo pojam parcijalne sume, naime kada se govori o Fourierovom redu sam pojam reda označava beskonačnu sumu, dok se suma s određenim brojem članova naziva parcijalna suma. Kada zbrojimo ta

dva člana dobivamo sljedeću funkciju: $S_2(t) = \frac{4}{\pi}(\sin(t) + \frac{1}{3}\sin(3t))$. Na slici 3. prikazana je originalna funkcija i parcijalna suma s dva člana [12] [13].




*Slika 3: Prikaz originalne funkcije i parcijalne sume s dva člana [autorski rad]*

Može se zaključiti da je sada aproksimacija bolja, ali još uvijek ne prati originalnu funkciju dovoljno dobro. Znači da je potrebno dodati još članova, novi članovi se dodaju na isti način, samo smanjujemo period i amplitudu te ih pribrajamo ostalima. Na slici 4 prikazana je parcijalna suma s 10 članova.




*Slika 4: Prikaz originalne funkcije i parcijalne sume s 10 članova [autorski rad]*

Vidi se da 10 članova daje vrlo dobru aproksimaciju. Kada bi nastavili dodavati članove, približavajući se beskonačnosti, aproksimacija bi postajala sve bolja. U praksi je potrebno naći balans između broja članova i točnosti aproksimacije što ovisi o izgledu originalne funkcije.

U ovom primjeru pokazano je što Fourierov red postiže, tj. kako se neku periodičku funkciju može zapisati kao sumu sinusoida. Ali ovaj primjer ne daje odgovor kako izračunati pojedine članove, zato će se u nastavku prikazati matematički zapis Fourierovog reda

$$f(t) = a_0 + \sum_{m=1}^{\infty} a_m \cos\left(\frac{2\pi m t}{T}\right) + \sum_{n=1}^{\infty} b_n \sin\left(\frac{2\pi n t}{T}\right)$$

i postupak za računanje članova reda za proizvoljnu periodičku funkciju [12] [13]. U izrazu se pojavljuju koeficijenti $a_0$, $a_m$ i $b_n$. Ti koeficijenti određuju „težinu“ pojedine funkcije sinusa i kosinusa za svaki izračunati član. Navedeni koeficijenti računaju se prema sljedećim formulama [7] [12]:

$$a_0 = \frac{1}{T}\int_0^T f(t)dt \quad a_m = \frac{2}{T}\int_0^T f(t)cos\left(\frac{2\pi mt}{T}\right)dt \quad b_n = \frac{2}{T}\int_0^T f(t)sin\left(\frac{2\pi nt}{T}\right)dt$$

Za primjer računanja navedenih koeficijenata, a zatim i Fourierovog reda uzet ćemo funkciju zvanu trokutasti val *(engl. triangle wave)*. Navedena funkcija će se zatim aproksimirati s parcijalnom sumom od 2, 4 i 6 članova.

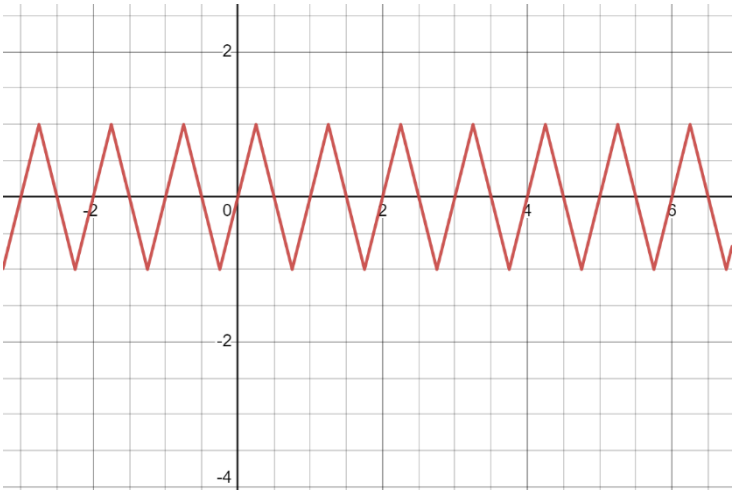
Trokutasta funkcija $f(t)$ s periodom $T$ i amplitudom $a$ zadana je sljedećom formulom [7]:

$$f(t) = \frac{2a}{\pi}\arcsin(sin\left(\frac{2\pi}{T}t\right))$$

Za primjer će poslužiti funkcija s periodom $T = 1$ i amplitudom $a = 1$. Kada se uvrste navedeni period i amplituda dobiva se sljedeća funkcija:

$$f(t) = \frac{2}{\pi}\arcsin(sin(2\pi t))$$

Na slici 5 prikazan je graf dobivene funkcije:




*Slika 5: Graf trokutastog vala [autorski rad]*

Za računanje Fourierovog reda najprije je potrebno izračunati koeficijente $a_0$, $a_m$ i $b_n$. Treba napomenuti to da je trokutasti val neparna funkcija. Funkcija je neparna ako vrijedi $f(-t) = f(t)$. Činjenica da je ovo neparna funkcija znači da ne sadrži kosinus članove, jer je kosinus parna funkcija. Zato računamo samo koeficijente

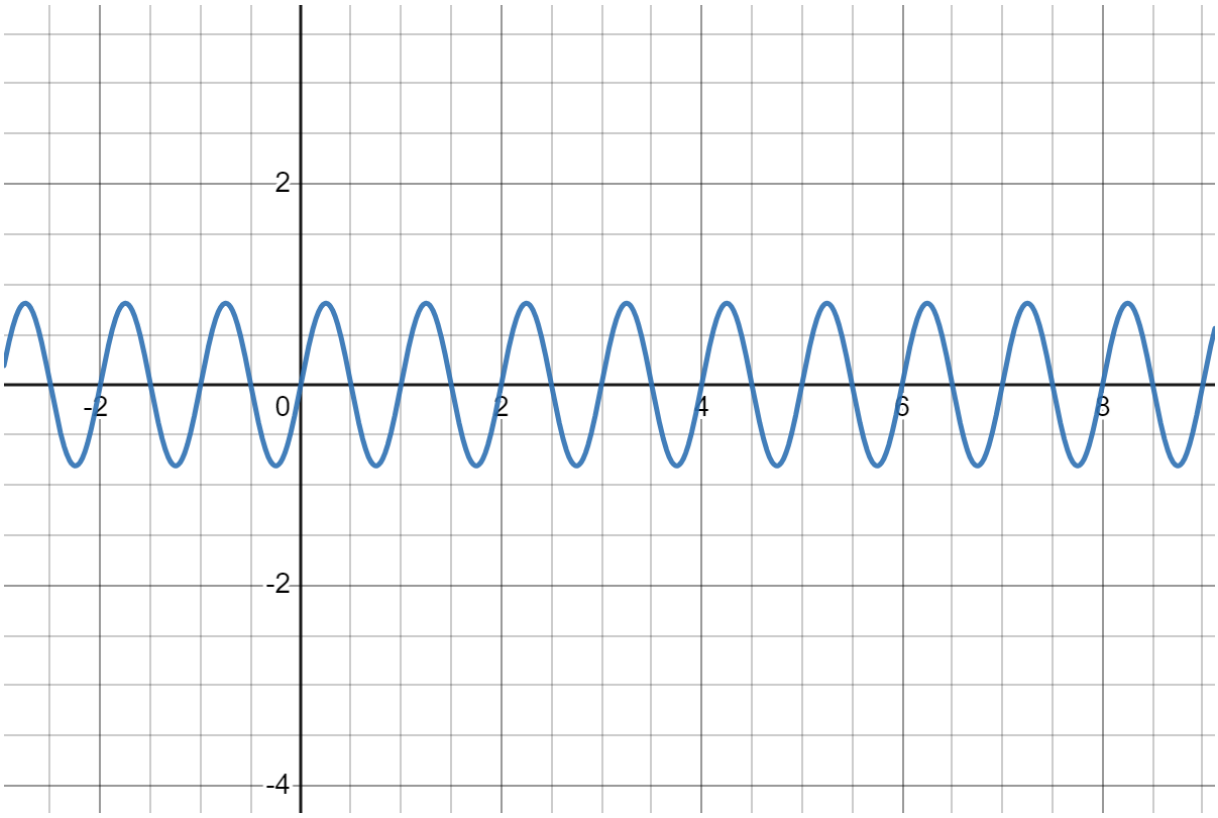
ispred sinusa $b_n$ i dobivamo izraz za računanje bilo kojeg $b_n$ koeficijenta za trokutast val:

$$b_n = \frac{8}{\pi^2}\frac{(-1)^{n-1}}{n^2}$$

Uvrštavajući $b_n$ dobivamo Fourierov red za gore prikazanu trokutastu funkciju:

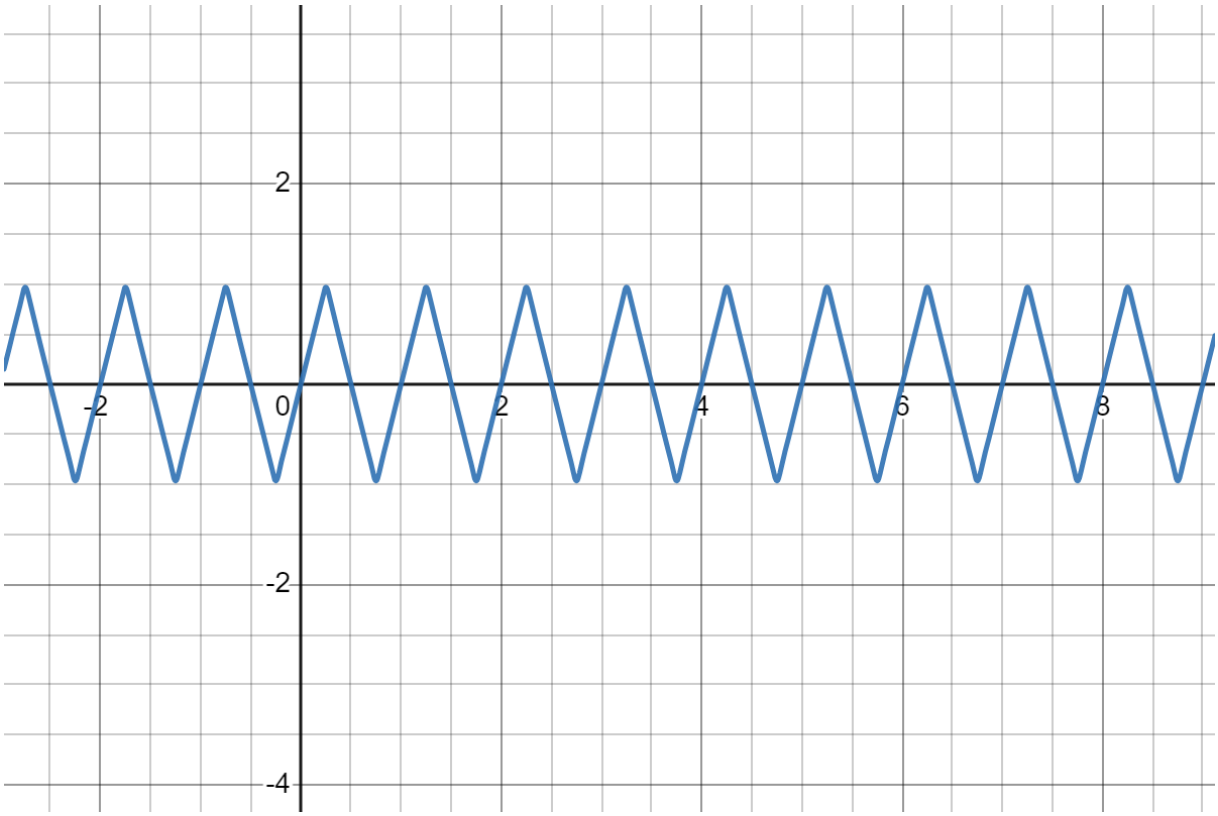
$$g(t) = -\frac{8}{\pi^2}\sum_{n=1}^{\infty}\frac{(-1)^n}{(2n-1)^2}\sin\left(2\pi(2n-1)t\right)$$

Na slici 6 prikazan je prvi član:




*Slika 6: Prikaz parcijalne sume s jednim članom [autorski rad]*

Iz slike se vidi da se već s jednim članom dobiva funkcija koja dosta dobro aproksimira originalnu trokutastu funkciju. Ako se izračuna parcijalna suma sa 6 članova dobiva se funkcija koja gotovo idealno aproksimira originalnu funkciju što je prikazano na slici 7:



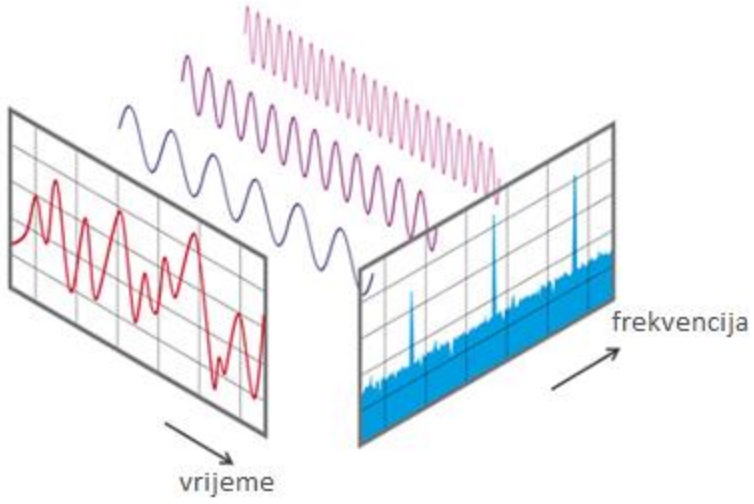

*Slika 7: Parcijalna suma sa 6 članova [autorski rad]*

Iz primjera koji su prikazani u ovom poglavlju vidi se da točnost aproksimacije koja se dobiva korištenjem parcijalnih suma ovisi o svojstvima originalne funkcije, npr. kada smo htjeli aproksimirati pravokutni val bilo je potrebno 10 ili više članova kako bi se dobila dovoljno dobra reprezentacija koja i dalje nije idealna, ali se kod trokutastog vala čak i s malim brojem članova dobiva vrlo vjerna reprezentacija.

## 3.3. Osnove Fourierovih transformacija

Sada kada je objašnjeno što je Fourierov red treba objasniti njegovu povezanost s Fourierovim transformacijama. Fourierova transformacija može se shvatiti kao proširenje Fourierovog reda. Za razliku od Fourierovog reda ona nije ograničena samo na prikaz periodičkih funkcija. Zbog toga Fourierove transformacije imaju potencijal za puno širu praktičnu primjenu jer kad god analiziramo neki stvarni signal (npr. zvučni zapis, podatke sa seizmografa) vrlo vjerojatno ćemo naići na signal koji nije periodički [7][8].

Fourierova transformacija signala (funkcije) je ekvivalentna, ali ipak bitno drugačija, reprezentacija tog signala (funkcije). Kao i Fourierov red, originalni signal se pretvara u oblik koji se može definirati kao suma sinusoida. Kod Fourierovih transformacija originalni prikaz signala naziva se vremenska ili prostorna domena ovisno o vrsti signala, a prikaz koji se dobije transformacijom naziva se frekvencijska domena. Pretvaranje signala u frekvencijsku domenu nudi mnoge prednosti, olakšava se analiza signala, otkriva se do tada skrivena struktura signala te se otkrivaju komponente signala s kojima je lakše računati nego s originalnim signalom. Na slici 8 prikazana je vizualizacija funkcije u vremenskoj domeni i njezine reprezentacije u frekvencijskoj domeni tj. njezine Fourierove transformacije [7][8][14].




*Slika 8: Vizualizacija funkcije u vremenskoj i frekvencijskoj domeni [15]*

Na slici se vidi originalna (crvena) funkcija koja nije ni parna ni neparna ni periodička, iza originalne funkcije prikazane su još tri funkcije čijim se zbrajanjem dobiva originalna funkcija. Plavi graf predstavlja vizualnu reprezentaciju Fourierove

transformacije, na x osi prikazana je frekvencija, na y osi prikazana je amplituda frekvencije. Mjesta gdje frekvencija „odskače“ od ostatka grafa predstavljaju komponente originalne funkcije tj. njihove frekvencije i amplitude. Iz slike se vidi osnovni koncept, ako imamo neku funkciju u vremenskoj ili prostornoj domeni njezina Fourierova transformacija daje nam spektar frekvencija iz kojeg se mogu iščitati frekvencijske komponente originalne funkcije.

Kao i kod Fourierovog reda postoji matematički način kako provesti Fourierovu transformaciju i dobiti spektar frekvencija koji u ovom slučaju predstavlja beskonačnu sumu sinusoida gdje svaka sinusoida ima neku frekvenciju i amplitudu. Krenimo od formule za Fourierovu transformaciju [16]:

$$F(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} f(t) e^{-i\omega t} dt$$

Već smo rekli da se Fourierova transformacija može smatrati proširenjem Fourierovog reda. Znamo da u Fourierov red možemo razviti samo periodičku funkciju, a u slučaju Fourierove transformacije period se razvlači na čitav brojevni pravac, tj. od $-\infty$ do $+\infty$. Fourierov red koristio je koeficijente $a_n$ i $b_n$, a u slučaju Fourierovih transformacija indeksi tih koeficijenata postaju kontinuirani dok su za Fourierov red to bili cijeli brojevi [17].

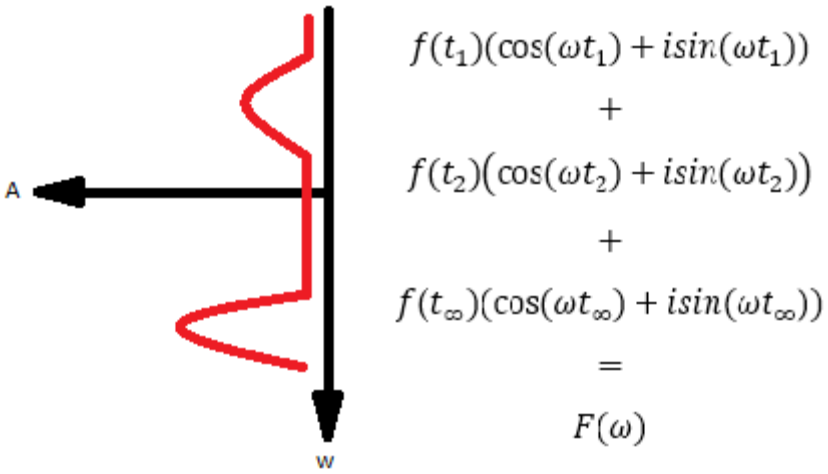
Gledajući formulu za Fourierovu transformaciju vidimo da je funkcija $f(t)$, koja je vremenska funkcija, reprezentirana pomoću druge funkcije $F(\omega)$ koja je funkcija iz frekvencijske domene.

Dakle Fourierova transformacija uzima funkciju iz vremenske domene i daje njezin ekvivalentni zapis u obliku funkcije iz frekvencijske domene. Dobivena funkcija u frekvencijskoj domeni je kompleksna tj. sadrži realni i imaginarni dio, izraz $e^{i\omega t}$ prema Eulerovoj formuli ekvivalentan je izrazu $\cos(\omega t) + i sin(\omega t)$. Dakle realni dio predstavlja funkciju kosinus na određenoj frekvenciji $\omega$, a imaginarni dio funkciju sinus na toj frekvenciji [16].

Kako bi razumjeli ulogu realnog i imaginarnog dijela tj. funkcija kosinus i sinus, možemo pogledati još jednu malo složeniju definiciju Fourierove transformacije koja kaže da je Fourierova transformacija uzima uzorak baziran na vremenu, mjeri svaki mogući ciklus i kao izlaz vraća amplitudu, pomak, i brzinu rotacije za svaki pronađeni

ciklus. U kontekstu ove definicije uzorak baziran na vremenu odnosi se na funkciju baziranu na vremenu, ciklus se odnosi na frekvenciju, fazni pomak (*engl. phase shift*) se odnosi na početne uvjete u funkciji a amplituda na amplitudu funkcije. Dakle Fourierova transformacija pretvara funkciju u frekvencijsku domenu te se iz frekvencijske domene mogu iščitati komponente originalne funkcije, svaka komponenta ima svoju frekvenciju, amplitudu i fazni pomak. Ova tri podatka za svaku komponentu dovoljno su dobra da se gotovo svaki signal/funkcija iz stvarnog svijeta prikaže kao suma sinusoida [14][16].

U nastavku je prikazan način kako se iz originalne funkcije dobiva funkcija u frekvencijskoj domeni:




*Slika 9: Način dobivanja funkcije u frekvencijskoj domeni [autorski rad]*

Gledajući odozgo prema dolje uzimaju se vrijednosti za $f(t)$, vrijednosti su kontinuirane od $-\infty$ do $+\infty$. Za svaku vrijednost računa se realni i imaginarni dio (kosinus i sinus), vrijednost se može smatrati frekvencijom za koju se ispituje da li je komponenta originalnog signala ili nije. Izračunata vrijednost za svaku pojedinu frekvenciju predstavlja amplitudu te frekvencije u originalnom signalu. Ako originalni signal sadrži tu frekvenciju to će se pokazati kao „skok“ u grafu koji je prikazan na lijevoj strani gornje slike. Graf mjeri amplitudu za svaku frekvenciju i na kraju „skokovi“ na grafu prikazuju koje su frekvencije komponente originalnog signala/funkcije. Također se vidi da je beskonačni zbroj sinusoida jednak Fourierovoj transformaciji $F(\omega)$ što nam i definicija Fourierove transformacije govori [18].

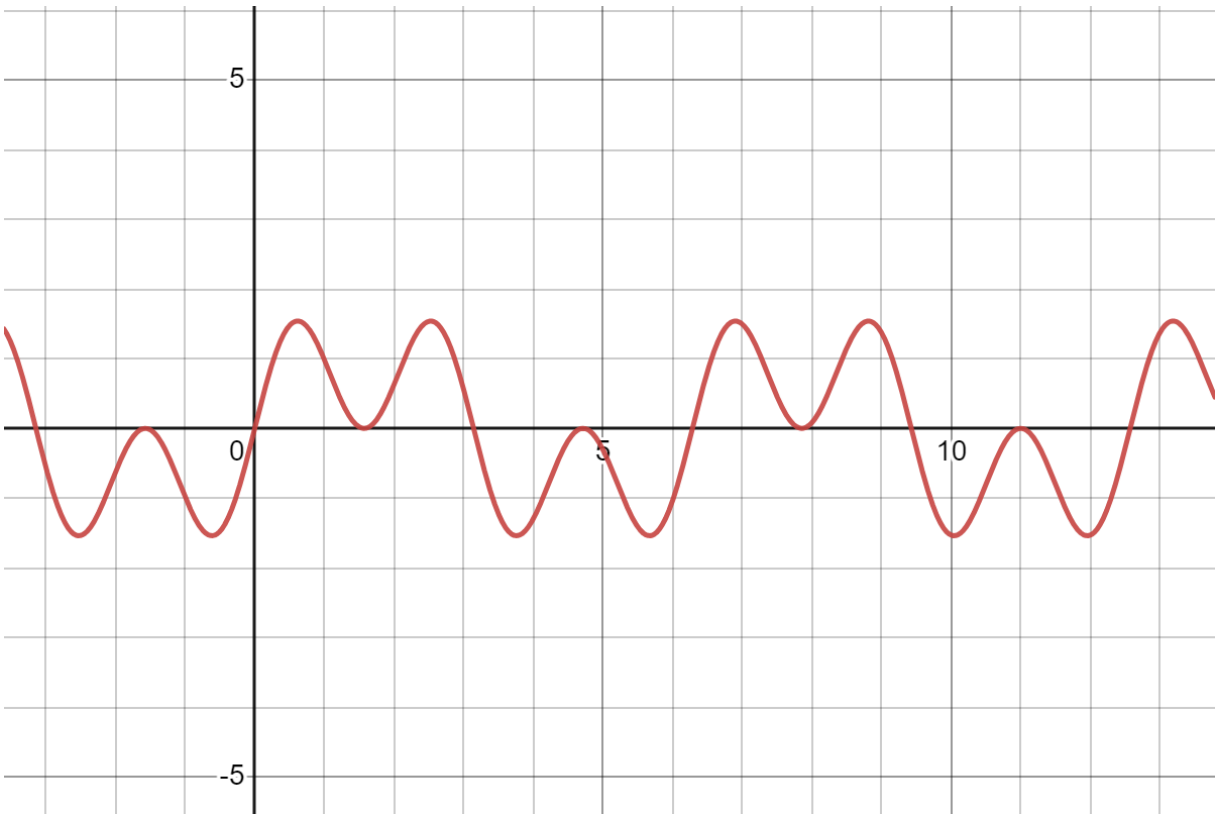
Još jedno svojstvo Fourierove transformacije koje nije do sad spomenuto je inverzna Fourierova transformacija (*engl. Inverse Fourier Transform*). Ono što se postiže inverznom transformacijom je dobivanje originalne funkcije $f(t)$ iz njezine ekvivalentne funkcije $F(\omega)$ koja je u frekvencijskoj domeni. U nastavku je prikazana formula za inverznu Fourierovu transformaciju [16]:

$$f(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} F(\omega) e^{i\omega t} d\omega$$

Iz formule se vidi da se inverznom transformacijom iz funkcije u frekvencijskoj domeni $F(\omega)$ dobiva originalna funkcija $f(t)$. Prednost postojanja inverzne transformacije jest mogućnost pohrane podataka o signalu/funkciji u frekvencijskoj domeni što u nekim slučajevima zahtjeva manje memorije, te analiza i manipulacija signalom u frekvencijskoj domeni što je često lakše nego rad s originalnim signalom.

## 3.4. Diskretne Fourierove transformacije (DFT)

U prijašnjem poglavlju objašnjeno je što je Fourierova transformacija, što se njome postiže i čemu služi. Prikazana je formula za transformaciju te je objašnjeno kako se transformacija računa. Pogledajmo sada kako bi se na praktičnom primjeru računala Fourierova transformacija. Recimo da imamo graf funkcije prikazane na slici 10:

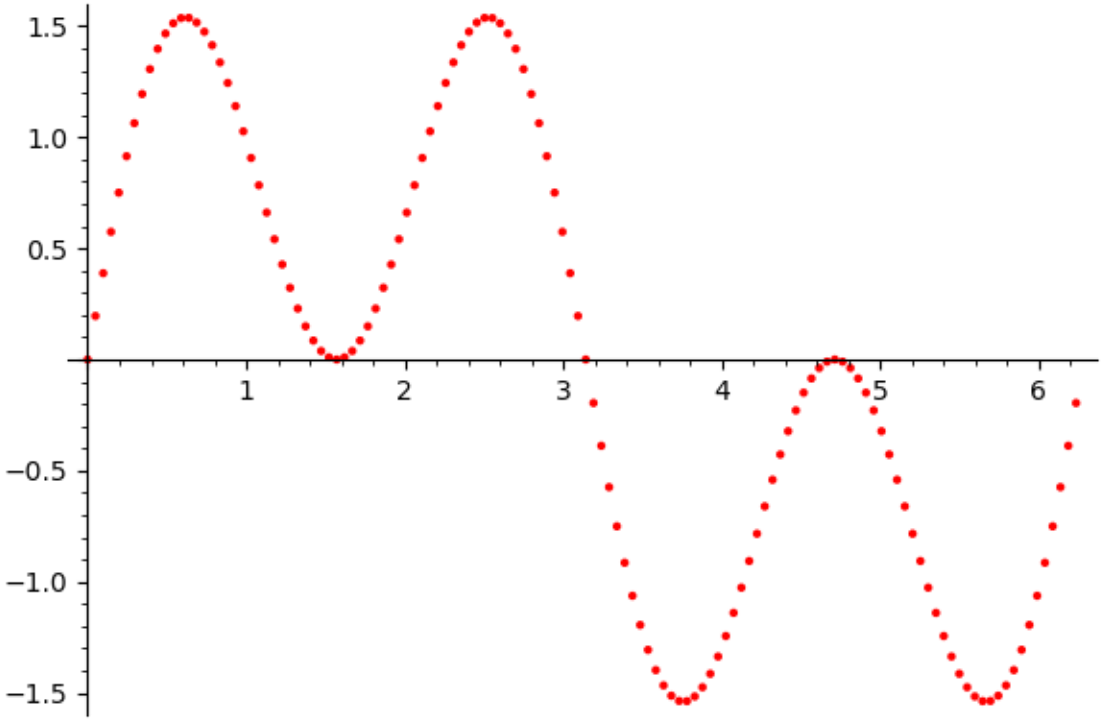



*Slika 10: Primjer grafa funkcije za računanje DFT-a [autorski rad]*

U ovom slučaju imamo samo graf funkcije ali ne znamo kako sama matematička reprezentacija funkcije izgleda ni koje su njezine komponente. Navedenu funkciju možemo smatrati signalom koji je snimio neki uređaj, ali svaki tako snimljeni signal neće snimati kontinuirane podatke već će postojati određena „zrnatost“, tj. uređaj snima vrijednost signala u određenim vremenskim intervalima. Vrijednost funkcije u određenoj točki u vremenu interpretira se kao amplituda te funkcije u to vrijeme. Dakle kada god se radi s nekim stvarnim signalom ne računa se Fourierova transformacija već diskretna Fourierova transformacija (DFT). Ulaz u DFT jest *N*

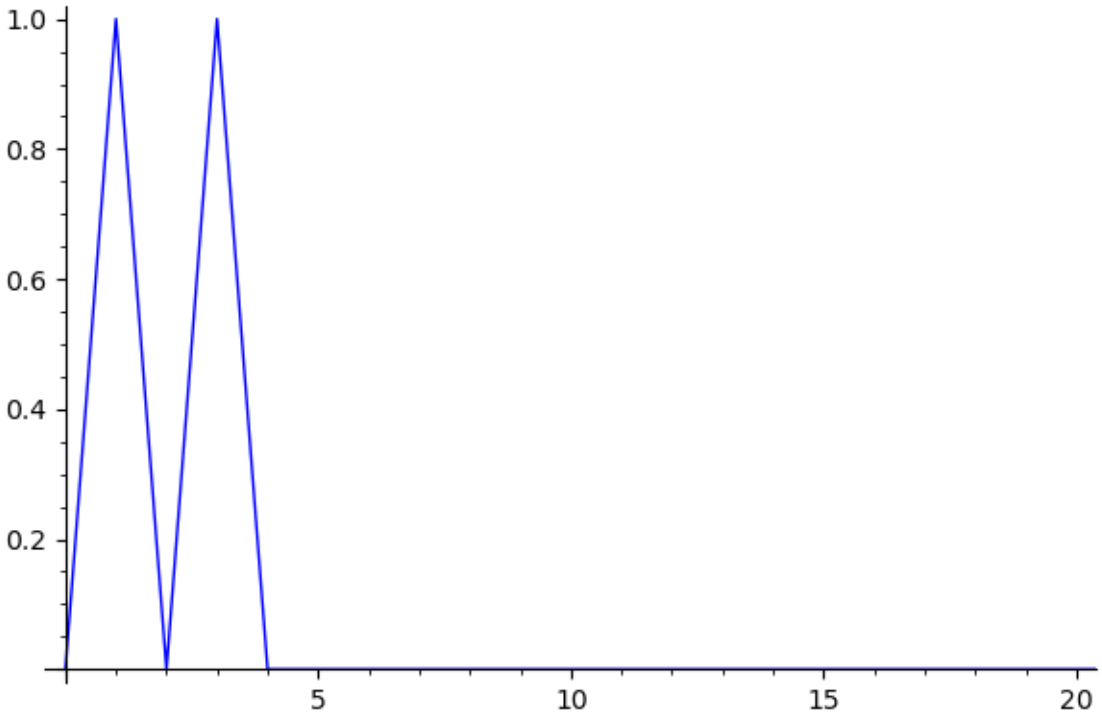
podataka koji predstavljaju vrijednosti zabilježenog signala u *N* vremenskih intervala [8][19].

Recimo da je za gornju funkciju zabilježeno 128 podataka, trajanje zabilježenog signala bilo je 6.28 sekundi, a podaci su bilježeni u jednakim vremenskim intervalima. Koristeći alat SageMath na slici 11 prikazani su zabilježeni podaci.




*Slika 11: Prikaz zabilježenih podataka [autorski rad]*

Ako se na navedenim podacima provede diskretna Fourierova transformacija, a dobiveni rezultati se prikažu na grafu na kojem x os prikazuje frekvencije a y os amplitude dobivaju se rezultati prikazani na slici 12:

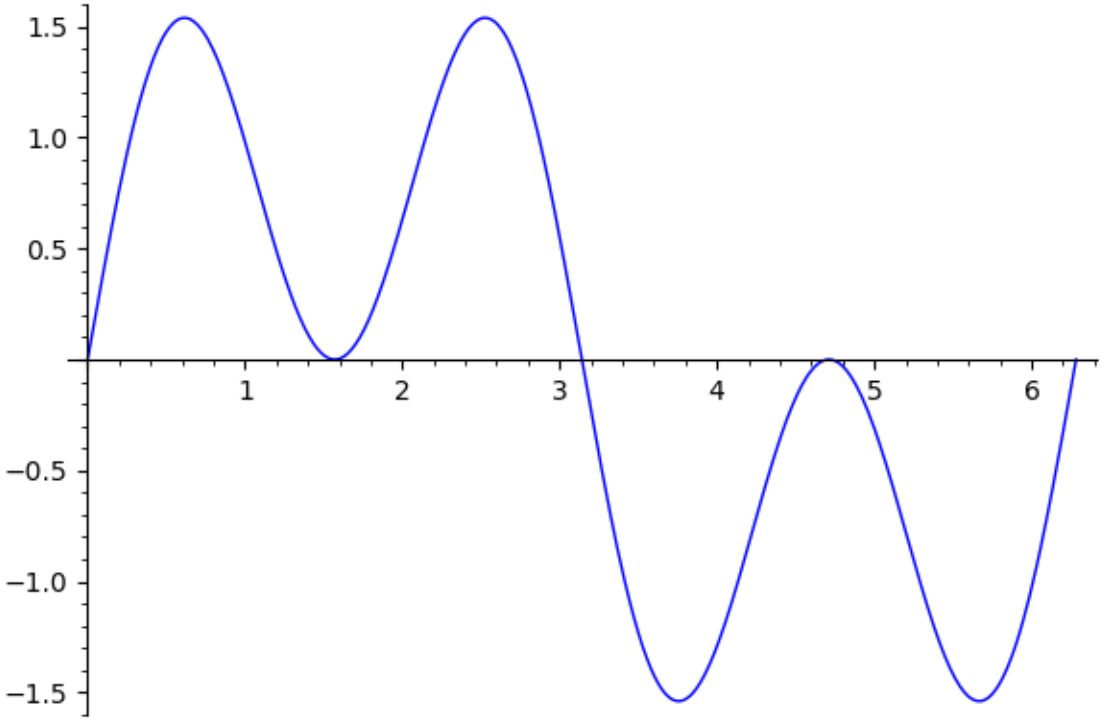



*Slika 12: Prikaz rezultata dobivenih diskretnom Fourierovim transformacijom [autorski rad]*

Iz grafa se vidi da originalni signal ima 2 komponente, jedna na frekvenciji 1 s amplitudom 1 i druga na frekvenciji 3 s amplitudom 1. Ako se dobivene komponente prikažu kao zbroj dvije sinus funkcije dobiva se sljedeća funkcija:

$$f(t) = \sin(t) + \sin(3t)$$

Kad se navedena funkcija nacrta na grafu, dobiva se originalna funkcija (slika 13):




*Slika 13: Prikaz zbroja komponenata dobivenih diskretnom Fourierovom transformacijom [autorski rad]*

Iako je funkcija korištena za ovaj primjer dosta jednostavna i sadrži samo dvije komponente može se vidjeti glavna ideja iza DFT-a. Iako nismo znali kako izgleda originalna funkcija tj. imali smo samo zabilježenih 128 podataka o vrijednostima funkcije u određenim vremenskim intervalima, korištenjem DFT-a uspješno smo dobili komponente originalne funkcije pomoću kojih je rekonstruirana originalna funkcija. Ovaj postupak radi s proizvoljno složenim funkcijama, provođenje transformacije na dovoljno dobrim podacima uvijek daje komponente originalne funkcije.

U nastavku će biti prikazan postupak računanja DFT-a na jednostavnoj funkciji s malom količinom zabilježenih podataka o vrijednostima te funkcije.

Za primjer uzimamo jednostavnu funkciju $f(t) = \sin(t)$ i uzimamo 8 vrijednosti te funkcije zabilježenih u jednakim vremenskim intervalima. Dobivamo sljedeće vrijednosti:

$$f_0 = \sin(0) = 0$$
$$f_1 = \sin\left(\frac{\pi}{4}\right) = 0.7071$$

$$f_2 = \sin\left(\frac{\pi}{2}\right) = 1$$

$$f_3 = \sin\left(\frac{3\pi}{4}\right) = 0.7071$$

$$f_4 = \sin(\pi) = 0$$

$$f_5 = \sin\left(\frac{5\pi}{4}\right) = -0.7071$$

$$f_6 = \sin\left(\frac{3\pi}{2}\right) = -1$$

$$f_7 = \sin\left(\frac{7\pi}{4}\right) = -0.7071$$

Formula za računanje DFT-a je sljedeća [20]:

$$F_k = \sum_{n=0}^{N-1} f_n e^{-i\pi kn/N}$$

Gdje je $N$ broj podataka s kojima radimo, a $k$ broj od 0 do $N-1$. U našem slučaju imamo 8 podataka ($N = 8$) pa će $k$ ići od 0 do 7. Za svaku od 8 izračunatih vrijednosti funkcije računamo sumu od $n = 0$ do $n = N-1$ po gornjoj formuli. Za svaku sumu dobivamo kompleksni broj kao rezultat. Rezultat predstavlja amplitudu određene frekvencije u originalnom signalu.

Broj podataka $N$ s kojima radimo jest broj rezultata koji ćemo dobiti transformacijom, svaki rezultat odnosi se na jednu frekvenciju, kreće se od $k = 0$ frekvencije do $k = N-1$ frekvencije, razlika između svake frekvencije određena je time koliko često mjerimo vrijednost signala, tj. koliki je razmak između $x$ koordinata. U našem slučaju interval je 1, pošto imamo 8 podataka imat ćemo 8 frekvencija tj. frekvencije od 0 do 7 herca. Treba napomenuti da broj podataka i trajanje signala utječe na točnost transformacije, što manje podataka imamo to je teže detektirati visoke frekvencije jer se u podacima ne vide nagle promjene vrijednosti signala. Isto tako što je manje trajanje signala to manje podataka o izgledu signala imamo te ćemo transformacijom dobiti netočne podatke, stoga je važno postići balans između količine podataka i trajanja signala [8][20].

U našem primjeru moramo izračunati sume od $k = 0$ do $k = 7$. Za potrebe računanja korišten je ekvivalentni zapis Eulerove formule [8][20]:

$$e^{it} = \cos(t) + isin(t)$$

Korištenjem ove ekvivalencije kao rezultat dobivamo kompleksan broj $F_k = A_k + B_k i$. U nastavku su prikazani izrazi za $k = 0$ i $k = 1$ te izračunati rezultati za ostale $k$ vrijednosti [20]:

$$F_0 = \sum_{n=0}^{7} f_n e^{-i2\pi \cdot 0 \cdot \frac{n}{8}} = \sum_{n=0}^{7} f_n \cdot 1 = 0 + 0.7071 + 1 + 0.7071 + 0 - 0.7071 - 1 - 0.7071$$

$$F_1 = \sum_{n=0}^{N-1} f_n e^{-i2\pi \cdot 1 \cdot \frac{n}{8}} = \sum_{n=0}^{N-1} f_n e^{-\frac{i\pi n}{4}} = 0 + 0.7071[\cos\left(-\frac{\pi}{4}\right) + i\sin\left(-\frac{\pi}{4}\right)] + 1[\cos\left(-\frac{\pi}{4}\right) + \text{isin}\left(-\frac{\pi}{4}\right)] + \cdots - 0.7071[\cos\left(-\frac{\pi}{4}\right) + i\sin\left(-\frac{\pi}{4}\right)] = 0 - 4i$$

$$F_2 = 0$$
$$F_3 = 0$$
$$F_4 = 0$$
$$F_5 = 0$$
$$F_6 = 0$$
$$F_7 = 0 + 4i$$

Iz rezultata vidimo da je $F_0 = 0$, ova vrijednost se uvijek odnosi na nultu frekvenciju i pokazuje za koliko je originalni signal pomaknut od $x$ osi, u ovom slučaju rezultat je 0 pa zaključujemo da originali signal nije pomaknut. Vrijednosti $F_1$ i $F_7$ jedine su koje nisu 0. Iz naše originalne funkcije očekivali bismo da imamo vrijednost s amplitudom 1 za frekvenciju $F_1$. Iz rezultata vidimo da $F_1$ pokazuje amplitudu od 4 što nije točno (amplituda se računa kao apsolutna vrijednost dobivenog kompleksnog broja), također vidimo da $F_7$ također pokazuje vrijednost što bi značilo da je frekvencija $F_7$ komponenta originalnog signala što također nije točno. Jedno od svojstava DFT-a jest da nije moguće detektirati frekvencije koje su veće od broja podataka izmjerenih u sekundi podijeljenih na pola. Navedeno svojstvo naziva se

Nyquistova frekvencija (*engl. Nyquist frequency*) nazvana prema Harryju Nyquistu, fizičaru zaslužnom za njeno otkriće. Broj podataka izmjerenih u sekundi (*engl. sampling frequency*) izražava se u hercima i u našem slučaju iznosi 8 herca. Prema tome u ovom primjeru nije moguće izmjeriti komponente veće od 4 herca te se te frekvencije, iako pokazuju neku vrijednost nejednaku nuli, ignoriraju. Pošto u našem slučaju ignoriramo pola podataka moramo sve frekvencije koje su pokazale neki rezultat udvostručiti pa tako za $F_1$ dobivamo vrijednost $0 - 8i$. Sada $F_1$ pokazuje amplitudu od 8 što i dalje nije točno, kako bi dobili točne podatke moramo izračunati prosjek za sve dobivene amplitude tako da podijelimo dobivene vrijednosti s brojem podataka, u našem slučaju dijelimo 8 s 8 i dobivamo 1, što je i očekivana amplituda[20][15].

Pošto smo kao originalnu funkciju uzeli sinus funkciju od 1 herca nije bilo teško interpretirati dobivene rezultate, ali što ako je početna funkcija bila pomaknuta za neku vrijednost, ili je bila kosinus funkcija, u tom slučaju jednostavnim čitanjem amplitude i frekvencije ne bi dobili dobre rezultate. Već je napomenuto da rezultat transformacije daje amplitudu, frekvenciju i pomak. Zato za dobiveni rezultat računamo pomak prema sljedećoj formuli [20]:

$$\theta = arctan\frac{B_k}{A_k}$$

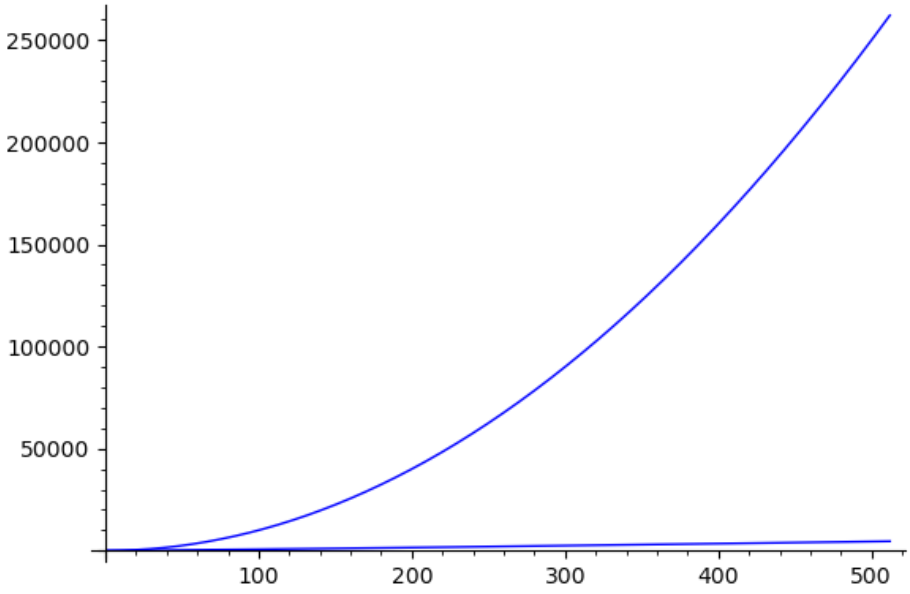
Kad bi uvrstili dobivenih $0 - 4i$ u formulu dobili bi dijeljenje s 0 što nije definirano, u tom slučaju kada je $A_k = 0$ pomak se određuje prema predznaku imaginarnog dijela, ako je predznak pozitivan pomak je $\frac{\pi}{2}$ a ako je negativan pomak je $-\frac{\pi}{2}$. U našem slučaju pomak je $-\frac{\pi}{2}$. Izračunati pomak se odnosi na kosinus funkciju, ako nacrtamo kosinus funkciju od 1 herca, s amplitudom od 1 i pomakom od $-\frac{\pi}{2}$ dobivamo originalnu funkciju i tako znamo da smo transformacijom dobili dobar rezultat.

Bez obzira od koliko komponenata je originalni signal sačinjen moguće ih je sve otkriti provođenjem DFT-a pod uvjetom da se odabere prikladan broj podataka i prikladan interval između bilježenja podataka.

## 3.5. Brze Fourierove transformacije (FFT)

Glavni problem kod računanja DFT-a za neki signal jest broj potrebnih izračuna. U prijašnjem poglavlju transformaciju smo računali kao sumu $N$ podataka te je za svaku sumu bilo potrebno provesti $N$ izračuna, drugim riječima ako imamo $N$ podataka potrebno je provesti $N^2$ izračuna. Za 8 podataka potrebno je provesti 64 izračuna što je bez problema izvedivo, ali u praktičnim primjenama Fourierovih transformacija brojevi podataka su puno veći. Ako imamo veliki set od milijun podataka potrebno je trilijun izračuna što u vrijeme kada su se DFT počele računati pomoću računala nije bilo moguće realizirati [8][15].

Kao što je već spomenuto u poglavlju o povijesti Fourierovih transformacija matematičar James Cooley pronašao je način kako puno brže računati DFT, točnije umjesto da je za $N$ podataka potrebno $N^2$ izračuna ovom metodom potrebno je $Nlog(N)$ izračuna. U prijevodu transformacija koja bi inače trajala godinama korištenjem ovog algoritma nazvanog FFT traje nekoliko minuta. U nastavku je prikazana razlika u rastu broja kalkulacija u slučaju diskretnih i brzih transformacija:



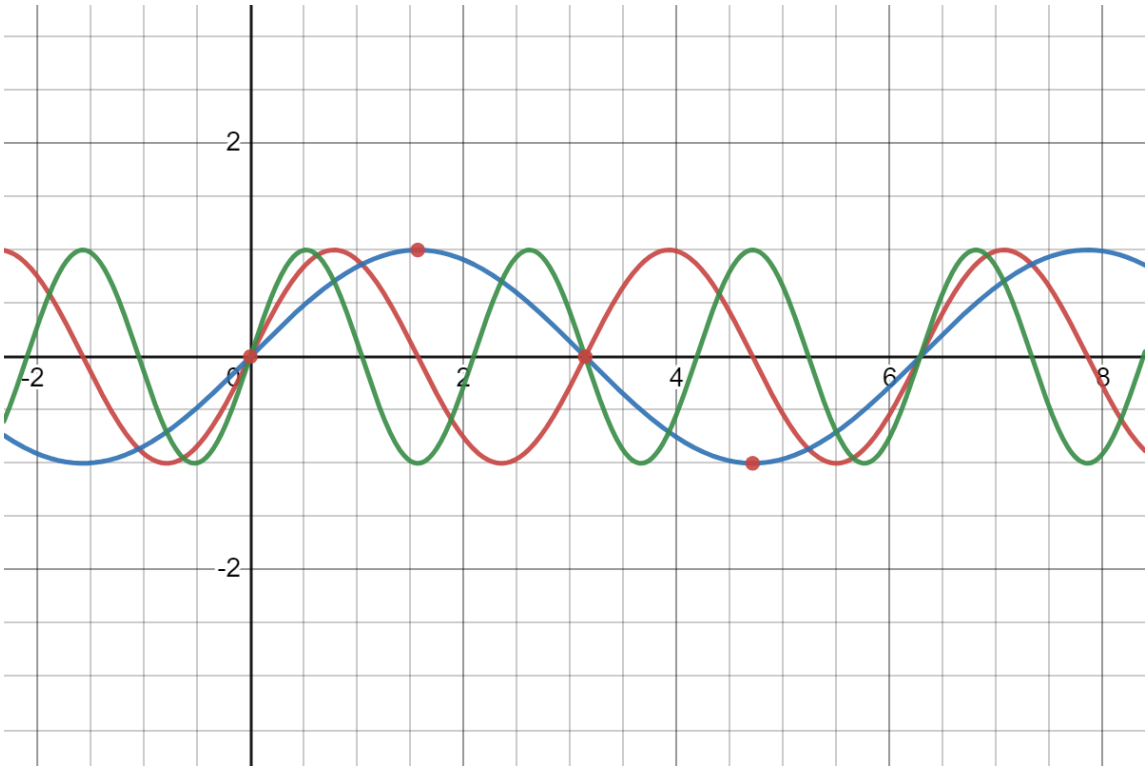

*Slika 14: Prikaz razlike u potrebnom broju izračuna za FFT i DFT [autorski rad]*

X os prikazuje broj podataka a y os prikazuje broj potrebnih izračuna, može se vidjeti ogromna razlika u broju potrebnih izračuna, razlika se povećava povećanjem broja podataka što znači da veći setovi podataka imaju više koristi od brzih Fourierovih transformacija. Ali kako od $N^2$ izračuna doći do $Nlog(N)$ izračuna?

Krenimo od ilustracije onoga što diskretna Fourierova transformacija radi s originalnom funkcijom, ako opet uzmemo funkciju $f(t) = \sin(t)$ i iz nje uzmemo 4 podatka dobivamo sljedeće vrijednosti:

$$f(t_0) = \sin(0) = 0$$
$$f(t_1) = \sin\left(\frac{\pi}{2}\right) = 1$$
$$f(t_2) = \sin(\pi) = 0$$
$$f(t_3) = \sin\left(\frac{3\pi}{2}\right) = -1$$

Naravno 4 podatka nisu dovoljna za provođenje transformacije ako želimo korisne rezultate, ali u ovom slučaju lakše je raditi sa samo 4 podatka za bolju vizualizaciju. Kada bi išli provoditi transformaciju za svaku pojedinu sumu množili bi podatke s tri sinus funkcije i tri kosinus funkcije kako bi dobili komponente originalne funkcije. Na sljedećoj slici prikazana su 4 izmjerena podatka i tri sinus funkcije koje se koriste u izračunu.




*Slika 15: Prikaz izmjerenih podataka i funkcija koje se koriste u izračunu [autorski rad]*

Iz slike se vidi da su podaci bilježeni od 0 do $2\pi$, kada bi radili diskretnu transformaciju za svaki podatak (crvena točka) bi morali napraviti 4 izračuna, pošto

ima 4 podatka radili bi ukupno 16 izračuna. FFT algoritam iskorištava činjenicu da su sinusoide periodične, tj. ako znamo vrijednost funkcije u $f(t)$ onda će ta ista funkcija imati tu istu vrijednost u $f(t+T)$ gdje je $T$ period funkcije. Na slici se može vidjeti da centralnu točku funkcije sijeku na mjestu gdje imaju istu vrijednost kao i u prvoj točki, također se vidi da funkcija u zadnjoj točki ima suprotnu vrijednost od vrijednosti u drugoj točki, to znači da možemo izračunati vrijednost transformacije za prve dvije točke i kopirati dobivene vrijednosti u druge dvije točke (i promijeniti predznak gdje je potrebno). Time smo smanjili potreban broj izračuna na $Nlog(N)$. U primjeru nisu pokazane kosinus funkcije, jer znamo da je originalna funkcija bila sinus pa kosinus funkcije neće korelirati s izmjerenim podacima U općenitom slučaju periodičnost funkcija može se primijeniti i na sinus i na kosinus funkcije [8].

Još ostaje pitanje kako za proizvoljni signal s proizvoljnim brojem podataka znati koje izračune treba odraditi, a koji se mogu samo kopirati. Ako pogledamo što smo radili u gornjem primjeru može se vidjeti da smo za one frekvencije koje su parne mogli samo prekopirati vrijednost (za $sin(0t)$ i $sin(2t)$) dok smo za neparne funkcije morali uzeti suprotnu vrijednost (za $sin(t)$ i $sin(3t)$). Ova podjela je zapravo zadnji korak FFT algoritma. Općenito ako imamo $N$ podataka i ako je $N$ potencija broja dva onda podatke možemo podijeliti na pola, jedna grupa će sadržavati parne podatke a druga neparne, za obje grupe možemo iskoristiti periodičnost sinus i kosinus funkcija te odraditi samo pola izračuna, a drugu polovicu prekopirati. Jednom podjelom upola smo smanjili broj potrebnih izračuna, ako nastavimo dijeliti podatke na pola i pri svakoj podjeli obavimo potrebni broj izračuna te ih prekopiramo doći ćemo do smanjenja potrebnog broja izračuna na željenih $Nlog(N)$ [8].

Opisani princip koristi se kad god se u praksi želi obaviti brza Fourierova transformacija, ali kao što je napomenuto ta brzina ovisi o tome je li broj ulaznih podataka potencija broja 2, ako to nije slučaj broj potrebnih izračuna se povećava i samim time se povećava potrebno vrijeme za obavljanje transformacije. Tako je na primjer potrebno više vremena ta se obavi transformacija za 200 podataka nego što je potrebno da se obavi za 256 podataka.

Otkriće algoritma za FFT otvorilo je vrata ogromnom broju primjena Fourierovih transformacija, danas gdje god ima ikakvog oblika obrade nekog signala vrlo je vjerojatno da se koristi FFT. FFT algoritam naišao je na široku primjenu i u polju računalne grafike te će se tim primjenama baviti ostatak ovoga rada.

## 3.6. Fourierove transformacije i računalna grafika

### 3.6.1. Uvod u računalnu grafiku

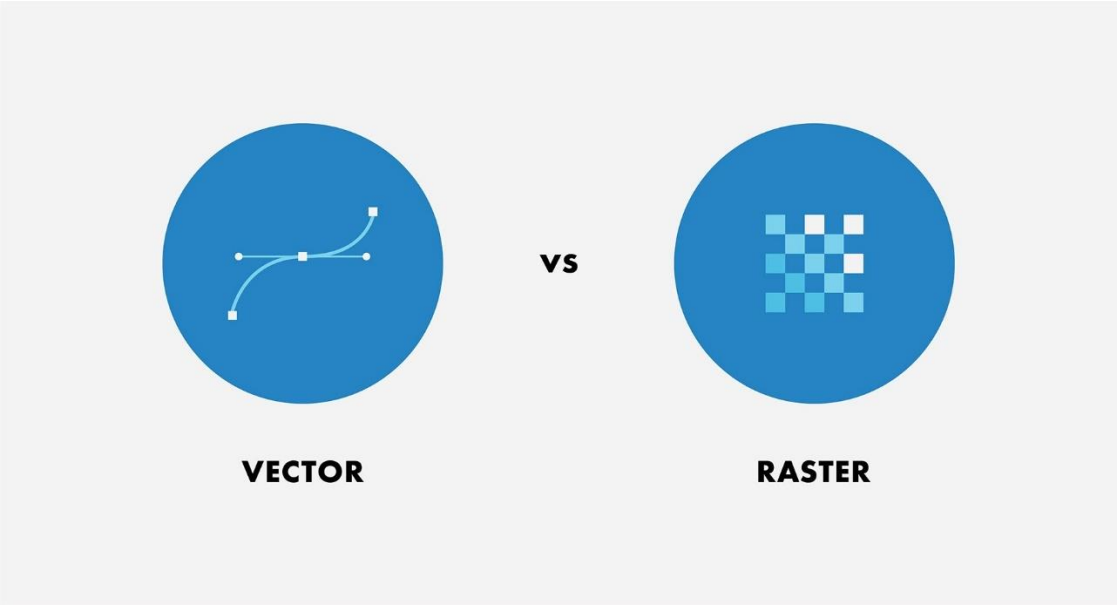
Kako bi razumjeli kako se Fourierove transformacije uklapaju u polje računalne grafike potrebno je razumjeti što je računalna grafika i što obuhvaća. Postoji mnogo definicija računalne grafike, jedna od njih govori da se računalna grafika odnosi na tehnologiju koja generira slike na zaslonu računala. Nešto općenitija definicija definira računalnu grafiku kao svaku skicu, crtež ili specijalnu mrežu koje slikovno prikazuju smislene informacije [21][22].

Općenito gledajući kada god se na računalu prikazuje neka slika koja je generirana pomoću računala ili snimljena digitalnom kamerom u njezin prikaz i generiranje je u uključena računalna grafika.

Polja računalne grafike i digitalne fotografije su usko povezana ali važno je napraviti razliku između njih. Računalna grafika općenito se odnosi na stvaranje i manipulaciju vizualnog sadržaja pomoću računala dok se digitalna fotografija odnosi na snimanje, obrada i spremanje podataka iz stvarnog svijeta u obliku digitalne fotografije. U procesu stvaranja vizualnog sadržaja polja računalne grafike i digitalne fotografije se isprepliću te su podjednako važna. Računalnu grafiku i digitalnu fotografiju stoga možemo smatrati poljima koja su neodvojiva kod stvaranja raznih vrsta digitalnog sadržaja, iz tog razloga će se u ostatku rada u kojem se govori o primjenama Fourierovih transformacija gledati primjene u računalnoj grafici i digitalnoj fotografiji [21][22].

Računalna grafika je svoje primjene našla u digitalnoj fotografiji, filmu i televiziji, video igrama i svim drugim elektroničkim uređajima koji služe za prikazivanje slika. Kada pogledamo široki spektar njezinih primjena možemo zaključiti da je u današnjem tehnološkom društvu računalna grafika jedno od najraširenijih polja u informatici i računarstvu [21].

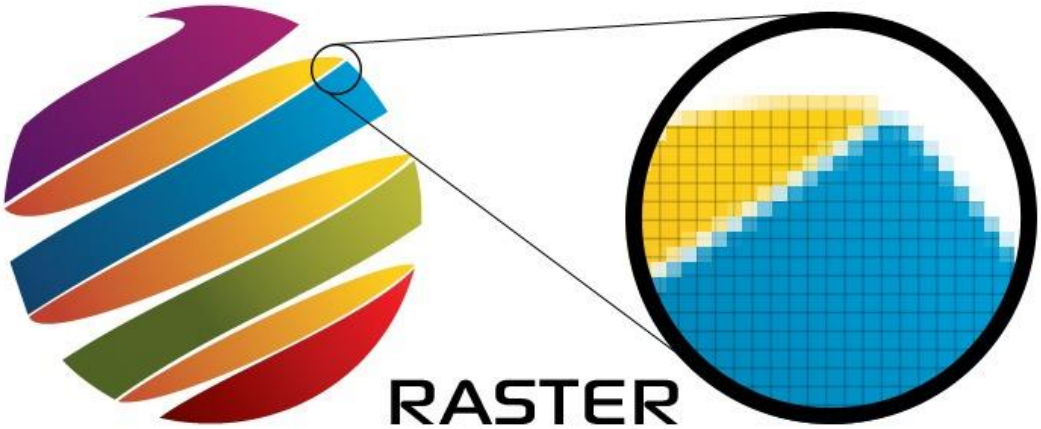
Za razumijevanje primjene Fourierovih transformacija u računalnoj grafici potrebno je objasniti razliku između dva osnovna tipa računalne grafike: rasterske grafike i vektorske grafike.




*Slika 16: Ilustracija koncepta vektorske i rasterske grafike [23]*

### 3.6.1.1. Rasterska i vektorska grafika

Rasterska grafika predstavlja najosnovniji i najintuitivniji oblik prikaza slike na računalu. Kod rasterske grafike slika se u računalu sprema u obliku dvodimenzionalnog polja u koje se spremaju podaci o pikselima. Prikaz slike je također dvodimenzionalno polje veličine koja odgovara rezoluciji slike, svaki element polja odgovara jednom pikselu određene boje [21].




*Slika 17: Ilustracija rasterske grafike [24]*

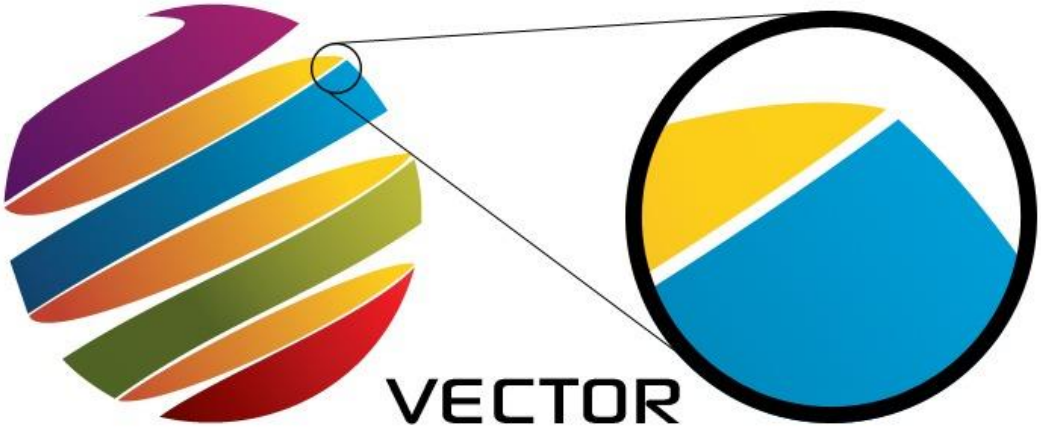
Najčešće se boja svakog piksela sprema u obliku RGB (*engl. red, green, blue*) vrijednosti. RGB vrijednost se sastoji od tri komponente od kojih svaka odgovara jednoj boji, konkretno crvenoj, zelenoj i plavoj. Za svaki piksel spremaju se tri numeričke vrijednosti od 0 do 255 koje govori o zastupljenosti pojedine boje u tom

pikselu. Neki formati slike koriste proširenje RGB vrijednosti u kojem se svakom pikselu uz RGB vrijednost dodaje vrijednost A (*engl. alpha*) koja predstavlja prozirnost tog piksela i izražava se vrijednostima između 0.0 i 1.0 [21].

U pravilu veći broj piksela znači veću rezoluciju slike i samim time oštriju sliku. Smanjenjem broja piksela slika postaje mutnija. Kod spremanja slika mora se postići balans između rezolucije i potrebne memorije za spremanje slika jer povećanjem rezolucije za faktor 2 (npr. slike od 128x128 piksela i 256x256 piksela ) za sobom povlače povećanje broja piksela za faktor 4 (od 16384 piksela na 65536 piksela). To znači da slika duplo veće rezolucije zahtjeva 4 puta više memorije za pohranu na računalu.

Za razliku od rasterske grafike vektorska grafika ne sprema podatke o pikselima. U slučaju vektorske grafike slike se u računalo spremaju u obliku geometrijski definiranih točaka, linija, krivulja i oblika. Kada korisnik želi pogledati tako definiranu sliku matematička (geometrijska) reprezentacija slike se pretvara u vizualnu reprezentaciju [21].

Prednost ovog način spremanja slike jest bolja kvaliteta slike i mogućnost skalabilnosti slike. Kada se rasterska slika povećava ili smanjuje pojavljuje se problem zamućivanja i izobličenja slike kao i pojave artefakta. Činjenica da su vektorske slike spremljene u obliku matematičkih formula znači da se mogu skalirati (povećavati i smanjivati) bez gubitka kvalitete, jer se pri svakom skaliranju slika ponovno generira koristeći matematički oblik u kojem je spremljena [21].




*Slika 18: Ilustracija vektorske grafike [24]*

Vektorske slike su gotovo uvijek generirane pomoću računala, a slika zabilježena kamerom se uvijek sastoji od piksela te ju nije moguće spremiti u vektorskom obliku.

U praksi primjene Fourierovih transformacija u vektorskoj grafici su gotovo nepostojeće i to iz mnogo razloga. Neki od njih su vrlo dobra optimizacija algoritama za stvaranje vektorske grafike, ne postojanje potrebe za kompresijom i izoštravanjem vektorske grafike (što je jedan od glavnih slučajeva korištenja Fourierovih transformacija) i činjenica da se vektorska grafika sastoji od oblika, linija i krivulja koje često nisu pogodne za Fourierove transformacije ili ih nema smisla transformirati.

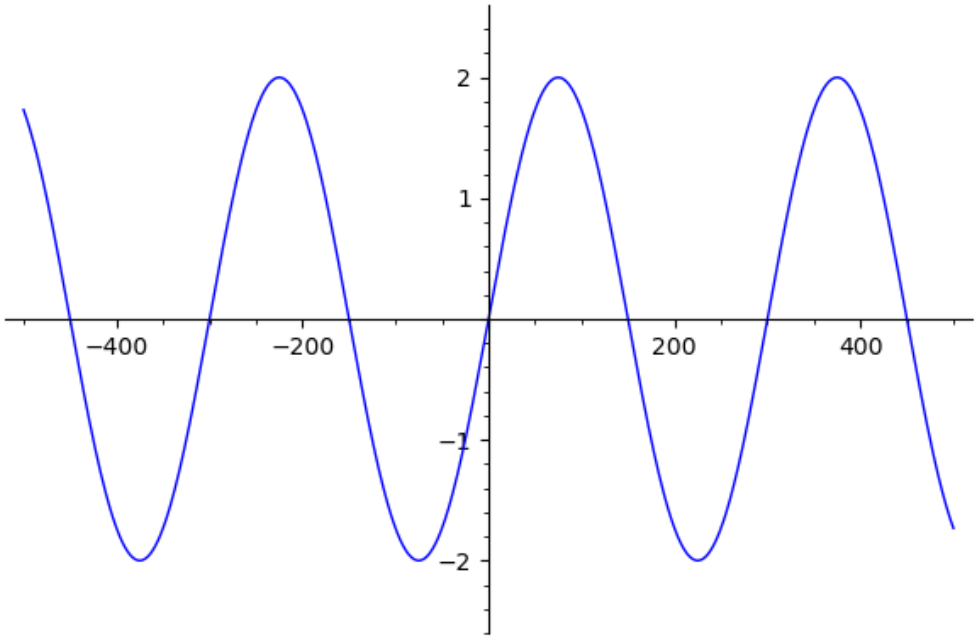
Gotovo svi slučajevi uporabe imaju veze s rasterskom grafikom pa ostatak ovog poglavlja nastoji objasniti kako se Fourierove transformacije mogu primijeniti na dvodimenzionalne grupe vrijednosti piksela i koji svi slučajevi korištenja proizlaze iz njihove primjene.

### 3.6.2. Povezanost Fourierovih transformacija i računalne grafike

Kako bi pokazali kako se bilo koja slika može prikazati kao suma sinusoida treba objasniti pojam sinusoidalne rešetke (*engl. sinusoidal grating*) [25].Općenito, sinusoida je određena s tri parametra: periodom ($T$), amplitudom ($A$) i faznim pomakom funkcije ($\varphi$):

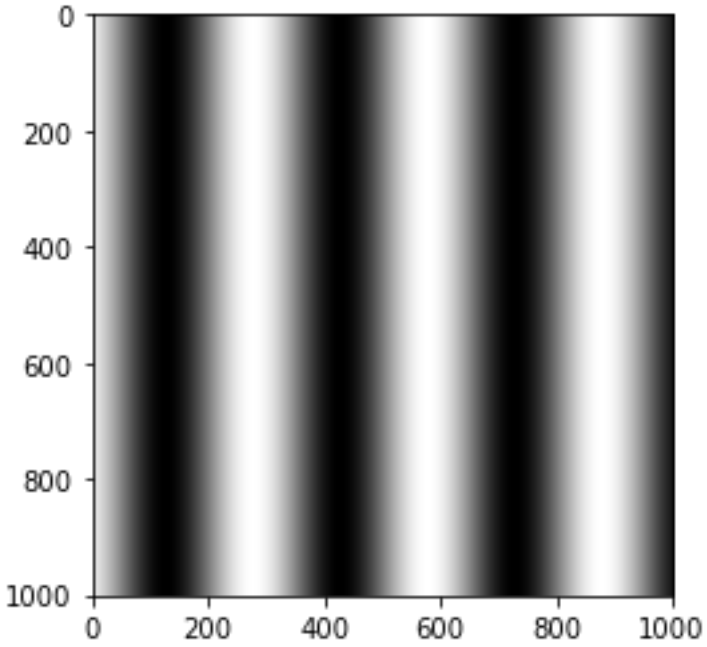
$$f(t) = A * sin(\frac{2\pi t}{T} + \emptyset)$$

Za primjer uzimamo funkciju sinus s amplitudom 2, periodom 300 i faznim pomakom 0. Navedena funkcija prikazana je na slici 19:



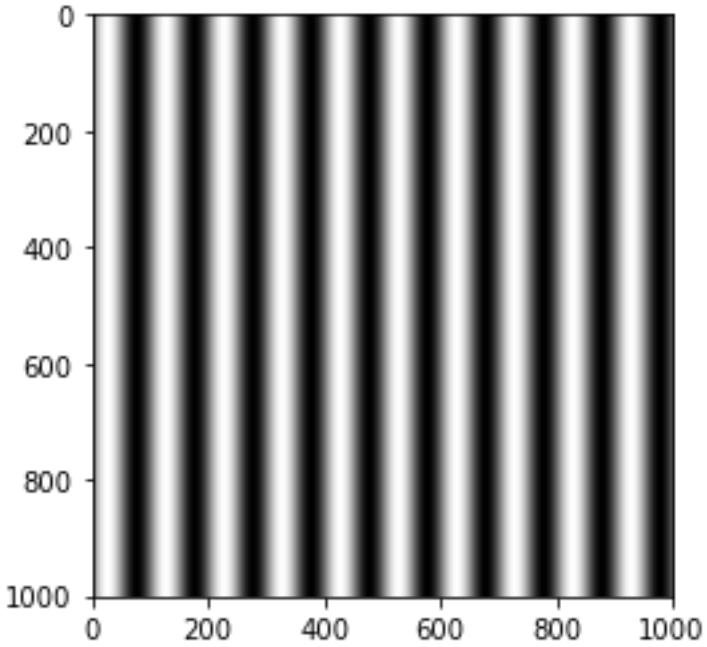

*Slika 19: Prikaz sinus funkcije s amplitudom 2 i periodom 300 [autorski rad]*

Sinusoidalna rešetka je dvodimenzionalna reprezentacija sinusoidalne funkcije u kojoj amplituda funkcije varira u određenom smjeru (orijentaciji). Sinusoidalna rešetka gornje funkcije (pri čemu je $t = x$) prikazana je na slici 20:




*Slika 20: Sinusoidalna rešetke funkcije s amplitudom 2 i periodom 300 [autorski rad]*

Što su „valovi“ gušći to je period manji i frekvencija veća, slika 21 prikazuje istu funkciju s periodom od 100:



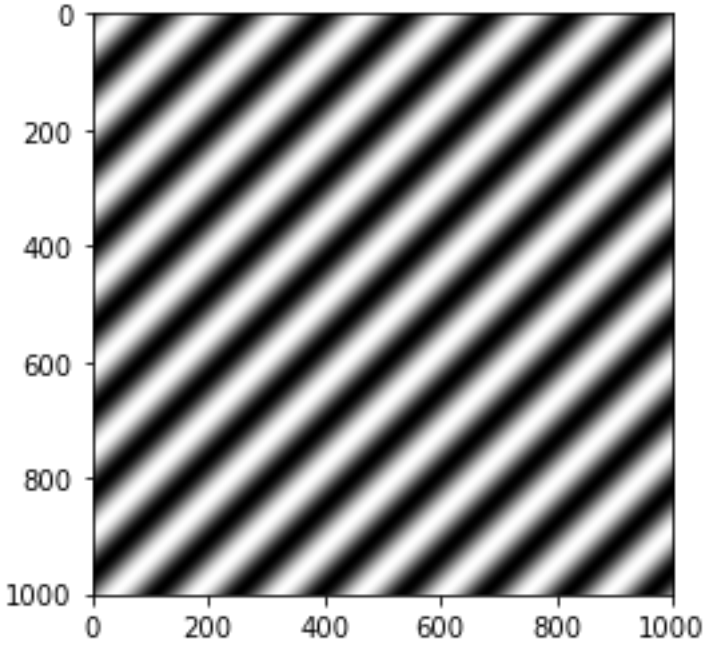

*Slika 21: Sinusoidalna rešetke funkcije s amplitudom 2 i periodom 100 [autorski rad]*

Orijentacija rešetke označava smjer pružanja valova na rešetki, rešetka se može rotirati prema sljedećoj formuli [25]:

$$t' = xcos(\phi) + ysin(\phi)$$

Gdje $\phi$ označava željeni kut rotacije u radijanima.

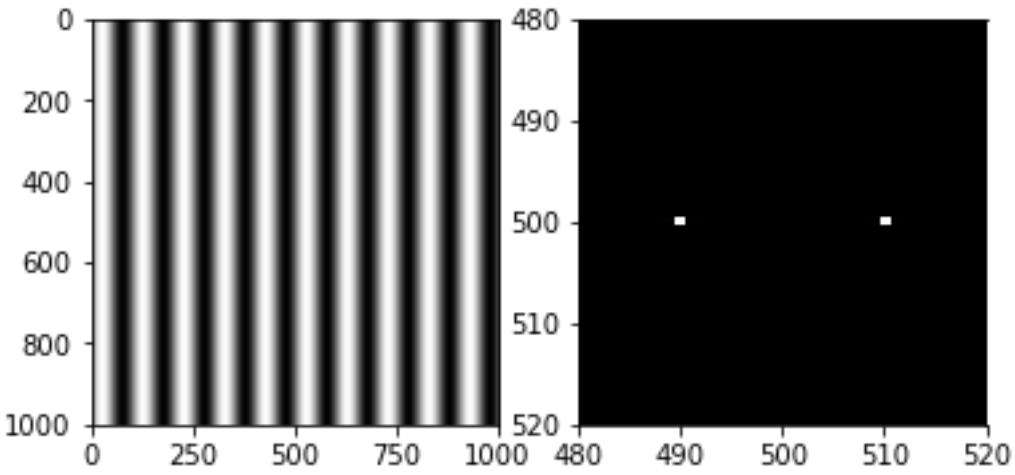
Ako prema ovoj formuli rotiramo gornju funkciju za $\frac{\pi}{4}$ radijana $(45^{\circ})$ dobivamo sljedeći rezultat:



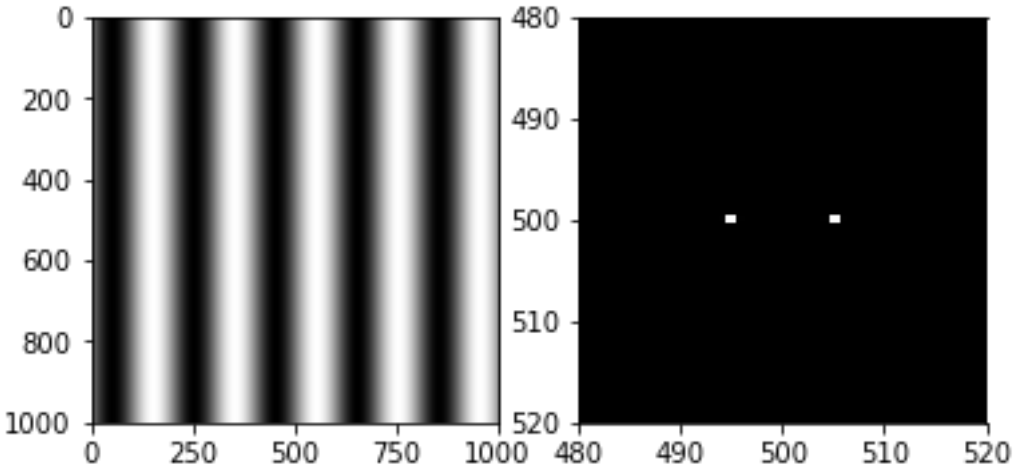

*Slika 22: Sinusoidalna rešetka rotirana za 45 stupnjeva [autorski rad]*

Sve rasterske slike mogu se dekonstruirati u sumu sinusoida različitih frekvencija, pomaka, amplituda i orijentacija sličnih onima prikazanima na gornjim primjerima. Zbrajanjem svih sinusoidalnih rešetki dobivenih dekonstrukcijom slike originalna slika može se rekonstruirati.

Pogledajmo sada rezultat izračuna FFT algoritma za gornje funkcije s periodom 100 i periodom 200:



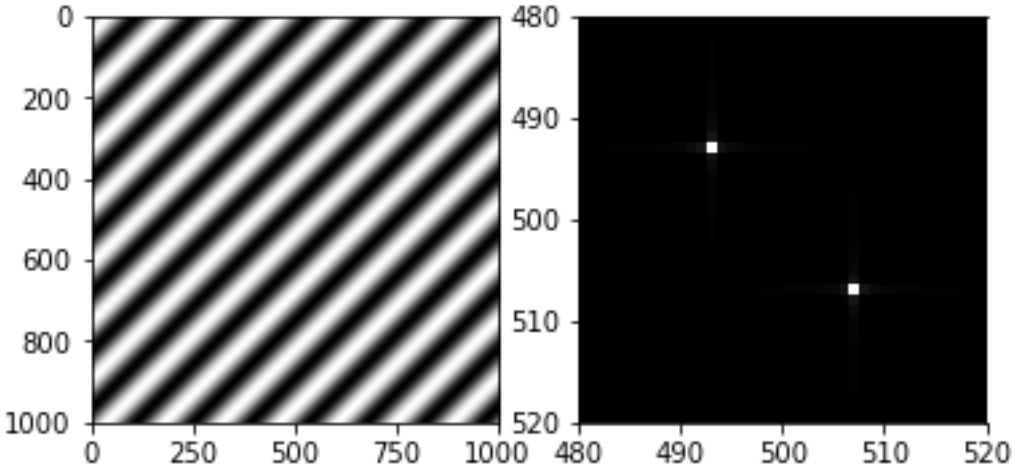

*Slika 23: Rezultat FFT-a za funkciju s periodom 100 [autorski rad]*




*Slika 24: Rezultat FFT-a za funkciju s periodom 300 [autorski rad]*

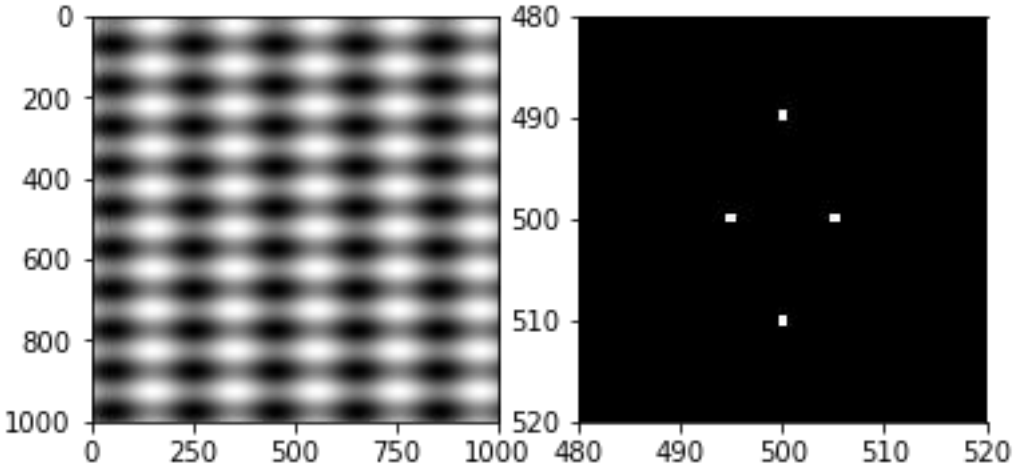
Kao što znamo iz poglavlja o diskretnim Fourierovim transformacijama rezultat transformacije je kompleksni broj, tj. ima imaginarni i realni dio. Rezultat transformacije prikazane na gornjim slikama također je kompleksni broj. Kako bi se rezultat transformacije prikazao na slici uzeta je apsolutna vrijednost realnog i imaginarnog dijela. Vidi se da obje transformacije prikazuju dvije točke, transformacije jedne sinusoide uvijek će se pokazati kao par točaka koje su centralno simetrične. Udaljenost točaka od centra označuje frekvenciju originalne funkcije, što su točke udaljenije od centra to je frekvencije veća tj. period je manji. Kut koji točke zatvaraju s horizontalnom linijom koja prolazi kroz središte predstavlja orijentaciju (rotaciju) originalne funkcije. Ovo svojstvo je prikazano na sljedećoj slici koja

prikazuje gornju funkciju s frekvencijom od 100 rotiranu za $45^{\circ}$ i njezinu transformaciju [25].




*Slika 25: Rezultat transformacije funkcije s frekvencijom 100 i rotacijom od 45 stupnjeva [autorski rad]*

Ako se napravi Fourierova transformacija na slici koja sadrži sumu dvije sinusoide očekivani rezultat bi bila transformacija koja sadrži 4 točke. Navedena transformacija prikazana je na slici 26:




*Slika 26: Rezultat transformacije dvije sinusoidalne funkcije [autorski rad]*

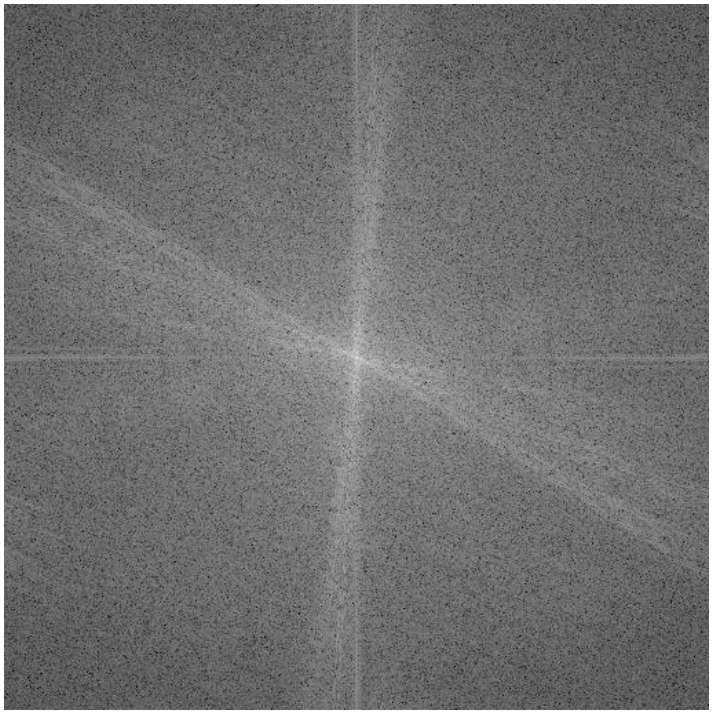
Gornja slika prikazuje sumu dvije sinus funkcije, jedna s frekvencijom od 100 rotirana za $90^{\circ}$ i druga s frekvencijom od 200. Rezultat pokazuje Fourierovu transformaciju na kojoj se vide parovi točaka koji se odnose na pojedine funkcije.

Dodavajući sinusoide na ovaj način u kombinacijama različitih frekvencija, pomaka, amplituda i rotacija može se dobiti svaka rasterska slika. Broj komponenti koje sadrži rasterska slika direktno ovisi o rezoluciji slike, slike veće rezolucije imaju

više komponenti, npr. slika rezolucije 301x301 piksel može se dekomponirati na 45300 sinusoida. Važan element za rekonstrukciju slike na sinusoide i uporabu Fourierove transformacije nad slikom jest uloga sinusoida s visokim i niskim frekvencijama. Niske frekvencije sadržavaju osnovne podatke o slici tj. slika s niskim frekvencijama bit će nedefinirana i mutna, visoke frekvencije daju slici strukturu, one definiraju oštre rubove tj. obrise slike. Slika s pretežno visokim frekvencijama sadržava samo obrise slike bez pozadinskih podataka [25].

Opisani način konstrukcije slike pomoću sinusoida ilustrira kako i zašto Fourierove transformacije rade na rasterskim slikama. Ako se svaka slika sastoji od sume sinusoida onda sliku možemo smatrati ulaznim signalom za brzu Fourierovu transformaciju, vrijednosti piksela predstavljaju set podataka s kojima radimo (možemo ga smatrati vrijednostima funkcije koju transformiramo). Broj podataka odgovara broju frekvencija s kojima množimo originalni set podataka. Kad se podaci o pikselima slike provedu kroz FFT algoritam dobiju se podaci o frekvencijama koje slika sadržava te o svojstvima tih frekvencija.

Na slici 27 prikazan izgled Fourierove transformacije za sliku rezolucije 512x512 piksela:



*Slika 27: Prikaz transformacije slike rezolucije 512x512 [autorski rad]*

Iz slike se vidi svojstvo centralne simetrije. Također se vidi raspored frekvencija u transformaciji. Centar transformacije sadrži podatke o sinusoidama na niskim frekvencijama, kako se udaljavamo od centra tako se povećava frekvencija sinusoida.

Rezultat FFT algoritma jest ekvivalentna reprezentacija slike u frekvencijskoj domeni, koja olakšava obradu slike na razne načine te optimizira spremanje slike u računalo.

Kada god se u polju računalne grafike koristi FFT algoritam za obradu slika koristi se ovaj princip pretvaranja slike u frekvencijsku domenu. U nastavku su prikazani glavni slučajevi primjene brzih Fourierovih transformacija u računalnoj grafici.

## 3.6.3. Primjena Fourierovih transformacija u računalnoj grafici

### 3.6.3.1. Prikaz slika

U prethodnom poglavlju objašnjeno je kako se svaka slika može rekonstruirati zbrajanjem određenog broja sinusoida s različitim svojstvima. Također je objašnjeno kako se navedeno svojstvo može iskoristiti za uporabu algoritma brzih Fourierovih transformacija kako bi se dobio prikaz slike u frekvencijskoj domeni.

Još ostaje za objasniti kako se brza Fourierova transformacija primjenjuje na 2D polje koje sadrži informacije o pikselima.

Pošto radimo s dvodimenzionalnim poljem koristimo formulu za dvodimenzionalnu Fourierovu transformaciju [26]:

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-i2\pi(ux+vy)}dx\,dy$$

Također se koristi formula za dvodimenzionalnu inverznu transformaciju [26]:

$$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v)e^{i2\pi(ux+vy)}du\,dv$$

Navedene formule treba pretvoriti u formule za diskretnu dvodimenzionalnu Fourierovu transformaciju kako bi ih mogli koristiti na podacima o pikselima slike.

Formule za diskretnu transformaciju i inverznu diskretnu transformaciju prikazane su u nastavku [27]:

$$F[k,l] = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f[m,n]e^{-i2\pi\left(\frac{k}{M}m+\frac{l}{N}n\right)}$$

$$f[m,n] = \sum_{k=0}^{M-1}\sum_{l=0}^{N-1} F[k,l]e^{i2\pi(\frac{k}{M}m+\frac{l}{N}n)}$$

U konkretnim slučajevima većina algoritama za FFT nad dvodimenzionalnim podacima radi na sljedeći način. Prvo se uzimaju vrijednosti piksela za svaki red piksela na slici, za svaki red provede se brza Fourierova transformacija, dobiveni rezultat prikazuje koje su frekvencijske komponente prisutne u pojedinom redu piksela. Dobivene rezultate spremamo u dvodimenzionalno polje i nad njima provodimo još jedan set brzih Fourierovih transformacija za svaki stupac. Rezultat je dvodimenzionalna Fourierova transformacija koja sadrži podatke o frekvencijskim komponentama cijele slike. Treba napomenuti da nije važno krećemo li prvo s transformaciju redova pa stupaca ili krećemo sa stupcima pa onda redovima, u oba slučaja dobivaju se isti rezultati [8].

Provođenjem transformacije na gore opisani način originalna reprezentacija slike se pretvara u reprezentaciju u frekvencijskoj domeni.

#### 3.6.3.2. Filtriranje slika

Filtriranje slika korištenjem brzih Fourierovih transformacija odnosi se na nekoliko konkretnih primjena: izoštravanje slika, zamućivanje slika i detektiranje rubova slika *(engl. edge detection).*

Primjena ovih metoda puno je lakša u frekvencijskoj domeni nego u prostornoj domeni (originalni prikaz slike). Kako bi razumjeli zašto je filtriranje slike lakše u frekvencijskoj domeni moramo dobro razumjeti što je rezultat dobiven transformacijom slike u frekvencijsku domenu. U nastavku je prikazana slika u prostornoj domeni i njezina reprezentacija u frekvencijskoj domeni:

*Slika 28: Prikaz slike i njene reprezentacije u frekvencijskoj domeni [autorski rad]*

Već znamo da frekvencijska domena daje podatke o svakoj sinusoidi koja je komponenta originalne slike, također znamo da što se više udaljavamo od centra u frekvencijskoj domeni to su frekvencije više.

Ako bi u slici bilo manje sinusoida na visokim frekvencijama slika bi bila mutnija, ako bi pak bilo manje podataka o niskim frekvencijama slika bi izgubila pozadinske podatke tj. ostali bi samo podaci o oštrim rubovima. Ako imamo sliku u prostornoj domeni nije nimalo lagano izdvojiti visoke ili niske frekvencije i izbaciti ih van, ali ako je slika transformirana u frekvencijsku domenu taj zadatak postaje puno lakši. Ako na primjer iz slike želimo izbaciti visoke frekvencije onda vrijednosti uz rubove postavimo na 0, ako pak želimo izbaciti niske frekvencije onda vrijednosti oko centra postavimo na 0. Znamo da se korištenjem inverzne Fourierove transformacije iz frekvencijske domene dobiva originalna slika. Ako iz frekvencijske domene dio podataka postavimo na 0 korištenjem inverzne Fourierove transformacije i dalje dobivamo originalnu sliku ali u filtriranom obliku.

- **Izoštravanje slika**

Izoštravanje slika u kontekstu korištenja brzih Fourierovih transformacija i frekvencijske domene odnosi se na pojačavanje uloge sinusoida visokih frekvencija i pojačanje kontrasta slike. Kako bi se postigao željeni efekt potrebno je na sliku u frekvencijskoj domeni primijeniti filter za izoštravanje. Ako želimo pojačati ulogu visokih frekvencija onda vrijednosti na rubu slike množimo s nekim faktorom većim od 1. Ako želimo postići veći kontrast onda moramo smanjiti ulogu dijela podataka o niskim frekvencijama kako bi se smanjila prosječna svjetlina slike a sačuvali podaci o niskim frekvencijama. Jedan od načina za konstruiranje takvog filtera je množenje vrijednosti u središtu s nekom vrijednosti manjom od 1 (ovisno o željenoj jačini kontrasta), zatim od središta do određene udaljenosti od centra interpoliramo između vrijednosti s kojom smo množili centar do maksimalne željene vrijednosti. Ovime postižemo da se udaljavanjem od centra postepeno povećava uloga visokih frekvencija i smanjuje uloga niskih frekvencija. Nakon interpolacije sve ostale vrijednosti na rubovima množimo s određenom maksimalnom vrijednosti [28].

Rezultat korištenja ovakvog filtera prikazan je na slici 29.

*Slika 29: Rezultat korištenja filtera za izoštravanje [autorski rad]*

Iz rezultata se vidi da se primjenom filtera na originalnu zamućenu sliku postiže efekt izoštravanja i efekt pojačanja kontrasta.

Ova vrsta filtriranja je korisna iz jednostavnog razloga da neke slike bolje izgledaju kad su izoštrene te se iz njih može izvući više informacija. Primjena se također nalazi kada je potrebno detektiranje rubova bez gubitka podataka o originalnoj slici.

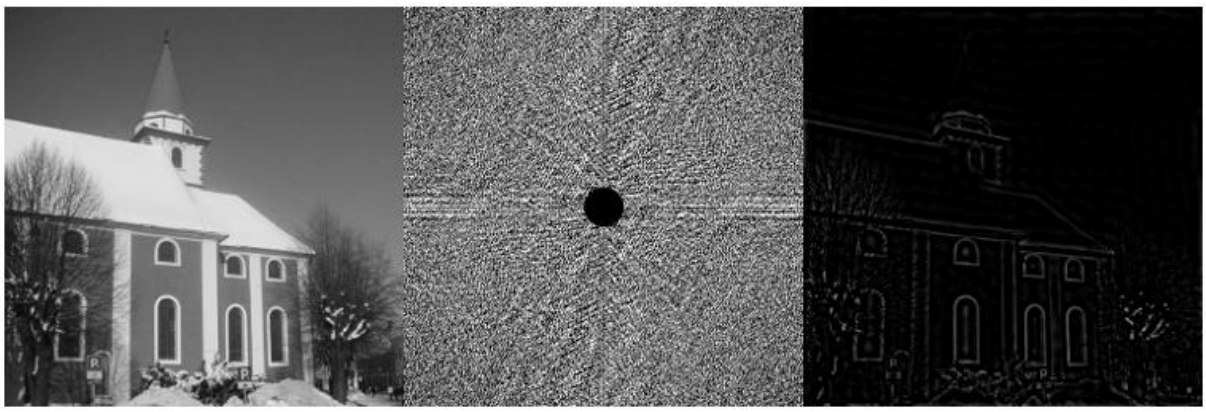
- **Detektiranje rubova**

Za detektiranje rubova koriste se visoko-frekvencijski *(engl. high pass*) filteri. Cilj ovih filtera je uklanjanje podataka o niskim frekvencijama i ostavljanje podataka o visokim frekvencijama, time se nakon inverzne transformacije dobiva slika koja sadrži samo obrise originalne slike.

Znamo da se kod standardnog FFT algoritma podaci o visokim frekvencijama spremaju na rubove dobivenog 2D polja. Ako se želi postići efekt visoko-frekvencijskog filtera potrebno je podatke od središta do željene udaljenosti od središta postaviti na nula, kako bi se kod inverzne transformacije koristili smo podaci o visokim frekvencijama.

Osnovni tip visoko-frekvencijskog filtera je tzv. idealni visoko-frekvencijski filter. Ovaj se filter konstruira točno po definiciji, određuje se željena udaljenost od središta (*engl. cut off distance*), vrijednosti koje su na udaljenosti manjoj od željene udaljenosti postavljaju se na 0 dok se vrijednosti na većoj udaljenosti ne diraju [28] [29].

Primjer originalne slike, primijenjenog idealnog visoko-frekvencijskog filtera na njenu Fourierovu transformaciju i rezultata prikazan je na slici 30.



*Slika 30: Prikaz rezultata primjene idealnog visoko-frekvencijskog filtera [autorski rad]*

Za gornji primjer korišten je visoko-frekvencijski filter u kojem su se vrijednosti na udaljenosti manjoj od 10% (od ukupno 100% koliko je do ruba) postavili na 0. U rezultatu vidimo obrise originalne slike osim obrisa tornja crkve koji se vrlo slabo vidi. Ovo je očekivani rezultat jer granica između tornja i neba na originalnoj slici nije oštro definirana tj. boje su im vrlo slične pa se u rezultatima ta granica manifestira u obliku pretežito niskih frekvencija koje ovaj filter postavlja na 0. rezultat također pokazuje pojavu tzv. Prstenastih artefakata (*engl. ringing*). Navedeni artefakti očitavaju se u obliku širenja obrisa oko svakog definiranog obrisa originalne slike te se zbog njih dobivaju nekvalitetni rezultati [28].

Prstenasti artefakti glavni su problem kod korištenja visoko-frekvencijskog filtera. Razlog njihove pojave jest „oštra“ granica na kojoj se vrijednosti prestaju postavljati na 0. Kako bi se pojava artefakata izbjegla potrebno je postići postepeni prijelaz između postavljanja vrijednosti na 0 i ostavljanja vrijednosti kakve jesu. Postoji nekoliko vrsta filtera koji postižu željeni efekt a najpopularniji od njih su Gaussov visoko-frekvencijski filter i Butterworthov visoko-frekvencijski filter [28][29].

Butterworthov filter se konstruira prema sljedećoj formuli [29]:

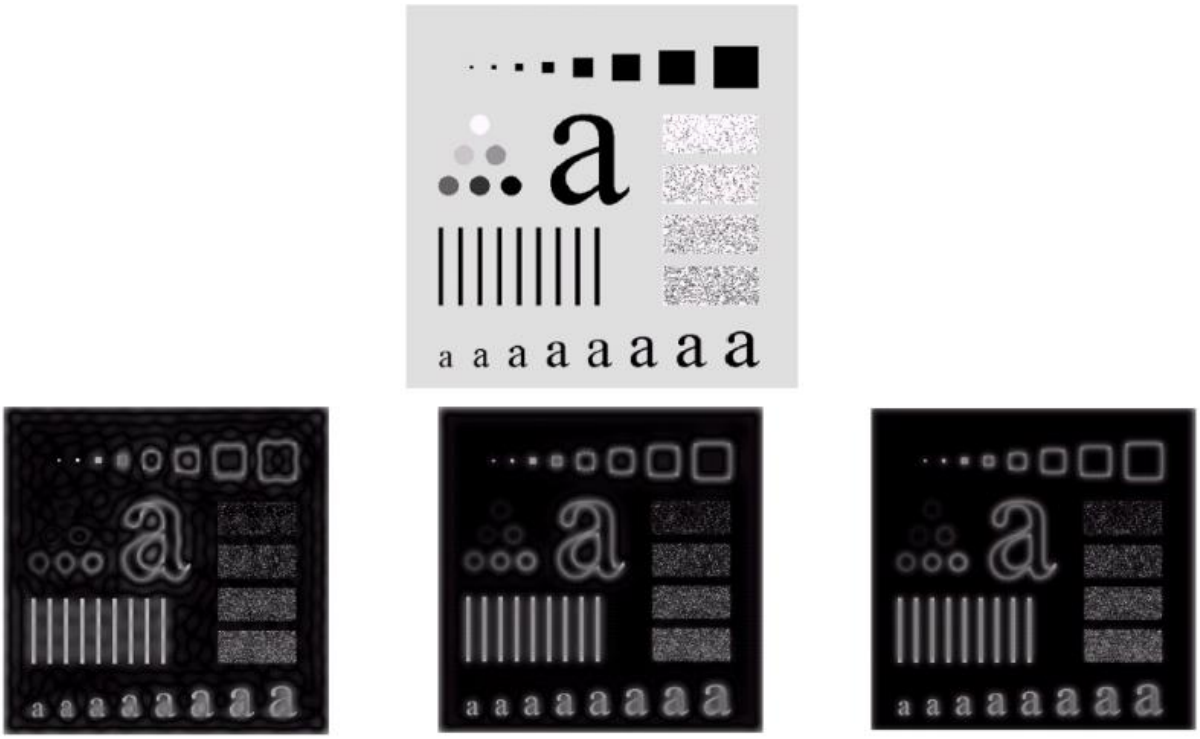
$$H(u,v) = \frac{1}{1 + [\frac{D_0}{D(u,v)}]^{2n}}$$

Gdje $u$ i $v$ označavaju položaj u frekvencijskoj domeni, $D(u,v)$ udaljenost od ishodišta frekvencijske domene, $n$ označava „jačinu“ filtera (*engl. order*) a $D_0$ željenu *„cut off“* udaljenost.

Gaussov visoko-frekvencijski filter konstruira se prema sljedećoj formuli [29]:

$$H(u,v) = 1 - e^{-D^2(u,v)/2D_0^2}$$

Oba filtera postižu slične rezultate i uspijevaju u postizanju željenog efekta uklanjanja prstenastih artefakata. Na slici 31 prikazana je originalna slika i efekt idealnog, Butterworthovog i Gaussovog visoko-frekvencijskog filtera s $D_0$ udaljenosti od 15.



*Slika 31: Efekt idealnog, Butterworthovog i Gaussovog visoko-frekvencijskog filtera [29]*

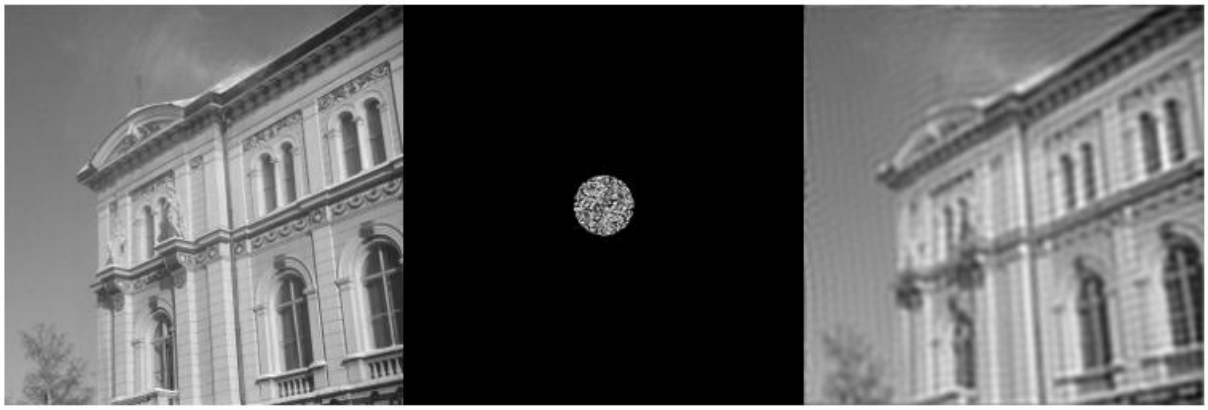
Visoko-frekvencijski filteri su primjenu našli u polju strojnog učenja (*engl. Machine Learning*)
i računalnog vida (*engl. computer vision*). Navedena polja se između ostalog bave analizom slika te detektiranjem (*engl. feature detection*) i izvlačenjem (*engl. feature extraction*) oblika iz njih. Pošto visoko-frekvencijski filteri naglašavaju rubove slika mogu poslužiti za navedene tehnike [30].

- **Zamućivanje slika**

Kako bi se postigao efekt zamućivanja slika potrebna je inverzija onoga što rade visoko-frekvencijski filteri, tj. potrebno je izostavljanje podataka o visokim frekvencijama i ostavljanje podataka o niskim frekvencijama. Vrsta filtera koji postižu efekt zamućivanja na navedeni način nazivaju se nisko-frekvencijski (*engl. low pass)* filteri [28][29].

Kod nisko-frekvencijskog filtera također se određuje željena „*cut off“* udaljenost te se vrijednosti veće od te udaljenosti postavljaju na 0 dok se vrijednosti manje od te udaljenosti ne diraju. Kod inverzne transformacije koriste se pretežito podaci o niskim frekvencijama i kao rezultat se dobiva zamućena slika bez oštrih i definiranih rubova [29].

Na slici 32 prikazan je efekt primjene nisko-frekvencijskog filtera.



*Slika 32: Efekt primjene nisko-frekvencijskog filtera [autorski rad]*

*„Cut off“* udaljenost je opet postavljena na 85 a u rezultatima se vidi željeni efekt zamućivanja. Kao i kod visoko-frekvencijskog filtera pojavljuju se prstenasti artefakti, razlog tome je opet oštra *„cut off“* udaljenost. Iz navedenog možemo zaključiti da je ovdje primijenjen idealni nisko-frekvencijski filtar. Rješavanje problema prstenastih artefakata jednako je kao i kod visoko-frekvencijskog filtera tj. potrebno je postepeno proći kroz željenu *„cut off“* udaljenost [29].

I u ovom slučaju postoji nekoliko načina kako u rezultatu ukloniti prstenaste artefakte, najpopularniji su kao i kod visoko-frekvencijskog filtera Butterworthov i Gaussov nisko-frekvencijski filtri.

Butterworthov nisko-frekvencijski filtar zadan je sljedećom formulom [29]:

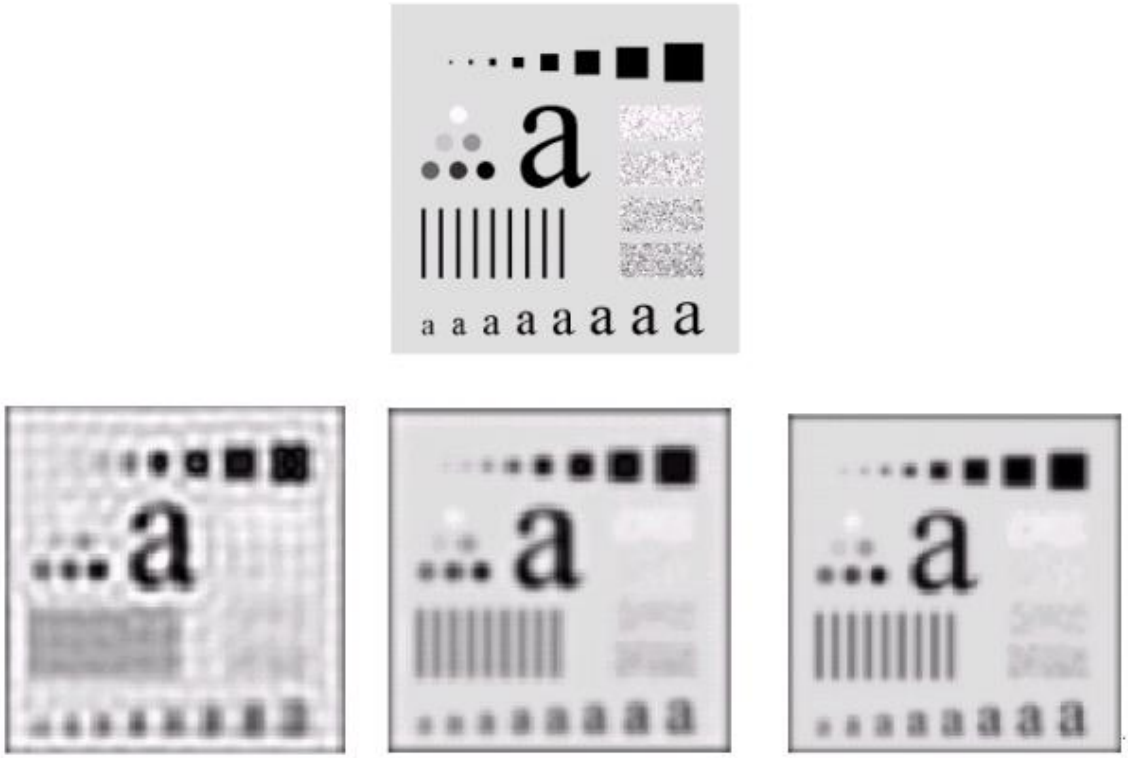
$$H(u,v) = \frac{1}{1 + [D(u,v)/D_0]^{2n}}$$

Gaussov nisko-frekvencijski filtar zadan je sljedećom formulom [29]:

$$H(u,v) = e^{-D^2(u,v)/2D_0^2}$$

Može se primijetiti da su formule za filtre gotovo identične, Gaussov filtar se računa na isti način samo što se kod visoko-frekvencijskog filtera izračunata vrijednost oduzima od 1. Kod Butterworthovog filtera razlika je jedino što se udaljenost neke prostorne koordinate od centra dijeli s *„cut off“* udaljenosti dok je kod visoko-frekvencijskog filtera postupak obrnut. Ova sličnost u formulama je očekivana jer je visoko-frekvencijski filtar zapravo inverzija nisko-frekvencijskog filtera.

Na slici 33 prikazana je originalna slika i efekt idealnog, Butterworthovog i Gaussovog nisko-frekvencijskog filtera s „*cut off*“ udaljenošću od 15.



*Slika 33: Efekt idealnog, Butterworthovog i Gaussovog nisko-frekvencijskog filtera [29]*

Nisko-frekvencijski filtri dobri su za filtriranje slika koje imaju jako „oštre obrise“ tj. kod slika koje su previše naglašene. Primjena se također vidi kod poboljšanja kvalitete zrnatih slika jer se nisko-frekvencijskim filterom postiže efekt izjednačavanja (*engl. smoothing*).

### 3.6.3.3. Kompresija i dekompresija slika

Jedna od općenito najraširenijih primjena brzih Fourierovih transformacija jest kompresija i dekompresija digitalnih fotografija. Važnost ove primjene posebno je naglašena danas kada je većina kamera na pametnim telefonima sposobna slikati slike visokih rezolucija.

Već je objašnjeno kako se provodi brza Fourierova transformacija na slici, prvo se transformiraju redovi pa stupci ili obrnuto, a rezultat daje reprezentaciju slike u frekvencijskoj domeni.

Krenimo od načina kako računala spremaju podatke o digitalnoj fotografiji. Moderni pametni telefoni mogu snimiti slike visoke rezolucije, za primjer ćemo uzeti da je slika rezolucije 3024x3024 piksela. Ovakva slika u sebi sadržava podatke o vrijednostima 9144576 piksela. Kada bi se podaci o vrijednostima svih tih piksela spremili na računalo to bi zahtijevalo puno memorije za samo jednu sliku. Konkretno spremanje slike rezolucije 3024x3024 u originalnom formatu s vrijednostima svih piksela zahtijeva približno 20 megabajta. 20 megabajta se za moderna računala na prvi pogled ne čini puno, ali lako se može vidjeti kako se količina potrebne memorije povećava ako znamo činjenicu da prosječni pametni telefon ima u sebi pohranjeno 2 do 3 tisuće slika. Veliki memorijski zahtjevi za spremanje slika u „izvornom“ formatu razlog su tome da gotovo svi najpopularniji formati slika koriste neki algoritam kompresije pri spremanju slike. Konkretno spremanje slike rezolucije 3024x3024 piksela u .jpg formatu zahtijeva oko 2 megabajta memorije. Iz navedenog se vidi velika redukcija u potrebnoj memoriji, a način na koji različiti formati postižu tu redukciju jest primjena algoritama za kompresiju koji se u većoj ili manjoj mjeri baziraju na Fourierovim transformacijama [8][28].

U praksi kada god se snimi digitalna slika nad njom se provodi brza Fourierova transformacija te se dobiva prikaz slike u frekvencijskoj domeni. Kao što znamo iz poglavlja o nisko-frekvencijskim filtrima većina vrijednosti u frekvencijskoj domeni koja se odnosi na visoke frekvencije može se postaviti na 0 bez da se značajno gubi na kvaliteti slike. Ovu činjenicu iskorištavaju algoritmi za kompresiju, iz slike visoke rezolucije može se odbaciti i više od 90% podataka o visokim frekvencijama te se spremaju samo preostali podaci o niskim frekvencijama. Ako to primijenimo na sliku rezolucije 3024x3024 piksela i iz nje izbacimo 90% frekvencija znači da umjesto da spremamo podatke o 9 milijuna moramo spremiti samo podatke o 900 tisuća frekvencija. Ovo smanjenje od 90 posto sa sobom povlači i redukciju potrebne memorije, navedeno se vidi u .jpg formatu (redukcija s 20 megabajta na 2 megabajta) [8][28].

Dakle kada god se sprema digitalna fotografija zapravo se sprema samo dio podataka iz frekvencijske domene. Kada takvu sliku želimo pogledati samo se provede inverzna Fourierova transformacija i dobiva se originalna slika u kompresiranom obliku [8][28].

Iz navedenog principa se vidi ogromna prednost brzih Fourierovih transformacija. Ako bi se transformirala slika rezolucije 3024x3024 pomoću DFT-a i ako bi jedan

izračun trajao 1 nanosekundu bilo bi potrebno gotovo jedan dan da se izračun obavi. Kad se koristi FFT algoritam isti izračun traje manje od sekunde.

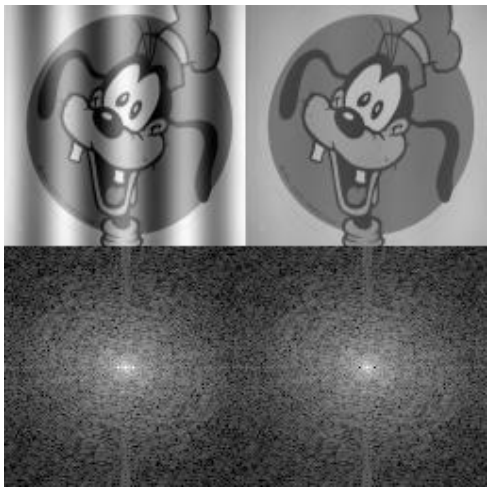
Isti princip se primjenjuje za inverznu transformaciju, koristeći brzu inverznu transformaciju vidi se velika razlika u vremenu izračuna.

#### 3.6.3.1. Kompresija i dekompresija videa

Princip opisan u prethodnom poglavlju može se jednostavno primijeniti na kompresiju i dekompresiju videa. Svi popularni formati spremanja videa na računalo koriste neki oblik kompresije koji se u većoj ili manjoj mjeri bazira na brzim Fourierovim transformacijama. Znamo da se svaki video sastoji od individualnih sličica (*engl. frames).* Svaka sličica predstavlja zasebnu digitalnu fotografiju koja se može kompresirati i dekompresirati na način objašnjen u prethodnom poglavlju. Vrijeme potrebno za kompresiju videa proporcionalno ovisi o trajanju videa, rezoluciji videa i broju sličica po sekundi (*engl. frames per second).*

#### 3.6.3.2. Uklanjanje artefakata

Artefakti na slikama često se nazivaju bukom (*engl. noise).* Pretvaranjem slike u frekvencijsku domenu postaje lakše izolirati artefakte te ih ukloniti, pogotovo ako znamo koje su frekvencije odgovorne za artefakte. Ovime se pri inverznoj transformaciji dobiva slika bez artefakata. Takav postupak uklanjanja artefakata dosta je teži u realnoj domeni. Na slici 34 prikazana je originalna slika koja ima buku u obliku jedne kosinus funkcije na određenoj frekvenciji. U frekvencijskoj domeni izolirana je navedena kosinus frekvencija i vrijednost joj je postavljena na 0, iz rezultata se vidi da je buka uklonjena a kvaliteta originalne slike očuvana [28].

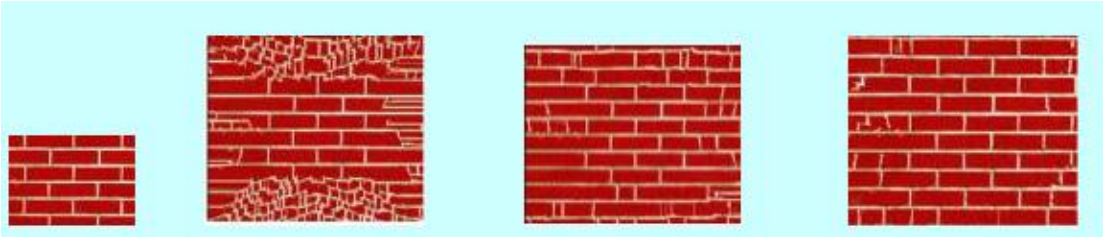


*Slika 34: Primjer uklanjanja buke sa slike [28]*

### 3.6.3.3. Ostale primjene u računalnoj grafici

Već je spomenuto da je računalna grafika raširena na sve oblike tehnologije dizajnirane za prikaz i kreiranje digitalnog sadržaja. Gdje god ima neke vrste analize i obrade signala vrlo je vjerojatno da FFT algoritam može olakšati i ubrzati neki korak obrade i analize. U prijašnjim poglavljima navedeni su i objašnjeni najrašireniji slučajevi korištenja brzih Fourierovih transformacija, ali treba znati da slučajeva korištenja ima još no oni se rjeđe koriste pa ih se neće detaljno prikazivati.

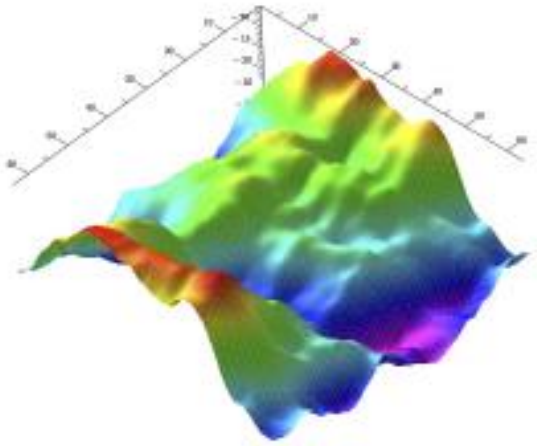
Jedan nešto rašireniji slučaj korištenja je generiranje tekstura pomoću brzih Fourierovih transformacija. Teksture se generiraju u različitim veličinama i oblicima na temelju ulazne teksture, čime se olakšava proces generiranja sličnih tekstura različite rezolucije i strukture. Također se može iskoristiti svojstvo uklanjanja artefakata prisutnih na postojećoj teksturi bez gubitka kvalitete teksture. Na slici 35 prikazani su primjeri tekstura generiranih na temelju ulazne teksture i korištenjem brzih Fourierovih transformacija [31].



*Slika 35: Prikaz generiranih tekstura [31]*

FFT se mogu iskoristiti i za generiranje 3D terena. Proces kreće od 2D reprezentacije signala koji sadrži širok spektar nasumičnih frekvencija (*engl. white noise*). Navedeni signal se pretvara u frekvencijsku domenu, na dobiveni rezultat

primjenjuje se filtar za skaliranje i provodi se inverzna transformacija. Rezultat se može iskoristiti za generiranje 3D terena. Na slici 36 prikazan je primjer terena generiranog pomoću brzih Fourierovih transformacija [32].



*Slika 36: Prikaz generiranog terena [32]*

# 4. Praktični dio

## 4.1. Korišteni alati

Pri izradi aplikacije za demonstraciju korištenja Fourierovih transformacija u obradi slika prvenstveno su korišteni alati za izradu web stranica jer je sama aplikacija izvedena u obliku web stranice.

Glavna arhitektura aplikacije izvedena je pomoću HTML-a, CSS-a i JavaScript programskog jezika, razvoj same stranice odrađen je pomoću Visual Studio Code aplikacije.

Za provođenje Fourierovih transformacija korištene su gotove biblioteka koje sadržavaju funkcije za izvođenje potrebnih brzih Fourierovih transformacija (FFT). Za tu svrhu odabrane su dvije biblioteke od kojih će se nakon testiranja koristiti ona brža. Prva odabrana biblioteka za izvođenje brzih Fourierovih transformacija je Mathjs, odabrana je zato što je jedna od popularnijih i pouzdanijih biblioteka za matematiku - Mathjs sadrži puno različitih funkcija koje omogućuju ugradnju širokog spektra matematičkih operacija unutar web stranica.

Konkretno unutar Mathjs biblioteke korištene su dvije funkcije koje obavljaju Fourierove transformacije i inverzne Fourierove transformacije tj. funkcije math.fft(arr) i math.ifft(arr) koje kao parametar primaju polje ili n-dimenzionalnu matricu.

Druga odabrana biblioteka nalazi se u sklopu „Project Nayuki“ web stranice. Stranica sadrži biblioteke za izvođenje brzih Fourierovih transformacija u različitim programskim jezicima pa tako i biblioteku za JavaScript koja je i korištena. Ova biblioteka je odabrana zato što se pokazala dosta brža od Mathjs biblioteke, sam kreator navodi da je u nju ugradio razne optimizacije kako bi poboljšao performanse i zato što je za njeno korištenje potrebno u web-stranicu uključiti samo jednu skriptu dok je za korištenje Mathjs biblioteke potrebno instalirati cijeli Mathjs.

## 4.2. Usporedba odabranih biblioteka za provođenje Fourierovih transformacija

Kao što je već navedeno za potrebe praktičnog rada odabrane su dvije biblioteke za izvođenje Fourierovih transformacija, te je u ovom poglavlju cilj kroz provođenje nekoliko testova odrediti koja je od njih brža i lakša za korištenje.

Kada se gleda lakoća korištena biblioteka rezultati su poprilično izjednačeni. Mathjs koristi funkcije *fft()* i *ifft()* za izvođenje transformacija, obje funkcije kao parametar primaju 2D polje što je prednost jer ovaj rad radi s podacima o pikselima koji su spremljeni u 2D polje. Nakon transformacije korištenjem Mathjs-a dobiva se ne-standardna struktura podataka te je potrebno paziti da se pri obradi dobivenih podataka ta struktura drži nepromijenjenom što je dosta otežavalo rad s ovom bibliotekom. Project Nayuki također koristi dvije funkcije (*transform()* i *inverseTransform()).* Obje funkcije kao parametar primaju jednodimenzionalno polje, što znači da je originalno 2D polje trebalo pretvoriti u 1D polje za obavljanje transformacije i onda opet u 2D polje za potrebe crtanja rezultata na canvas što otežava rad s bibliotekom. Struktura podataka koja se dobiva nakon transformacije je obično JavaScript polje što je prednost ove biblioteke.

Kada se uspoređuju brzine pokazalo se da je Mathjs biblioteka nekoliko puta sporija od Project Nayuki biblioteke. Konkretan razlog zašto je tome tako nije pronađen tijekom izrade ovog rada pa se može samo nagađati da je najvjerojatniji razlog to što je Project Nayuki puno bolje optimiziran od Mathjs biblioteke. U tablici 1 prikazana je usporedba brzine obavljanja transformacija između Mathjs biblioteke i Project Nayuki biblioteke za slike raznih rezolucija:

*Tablica 1: Usporedba odabranih biblioteka za provođenje Fourierovih transformacija*

| Rezolucija slike | Mathjs | Project Nayuki |
|---|---|---|
| **128x128** | 751.4 ms | 164.6 ms |
| **256x256** | 2520.3 ms | 281.5 ms |
| **512x512** | 10037.4 ms | 540.7 ms |
| **1024x1024** | 45902.7 ms | 1831.4 ms |

Iz tablice se vidi da je razlika u brzini vrlo značajna i samo se povećava kako se povećava rezolucija slike (broj piksela). S obzirom na dobivene rezultate odlučeno je da će se za praktični rad koristiti Project Nayuki.

## 4.3. Opis aplikacije za demonstraciju korištenja Fourierovih transformacija u obradi slika

Kao što je već prije spomenuto cilj izrade ove aplikacije nije bilo implementirati algoritam kojim bi se provele Fourierove transformacije unutar aplikacije već pokazati kako se Fourierove transformacije mogu iskoristiti za neku konkretnu svrhu unutar polja računalne grafike. Iz tog razloga korištena je Project Nayuki biblioteka tj. njezine funkcije koje omogućuju izvođenje brzih Fourierovih transformacija.

U teorijskom dijelu ovoga rada opisani su mnogi slučajevi korištenja Fourierovih transformacija u polju računalne grafike i izvan nje. Slučaj korištenja koji je odabran za ovu aplikaciju jest obrada slika, tj. cilj je bio razviti aplikaciju koja će omogućiti učitavanje slika u različitim formatima, izvlačenje podataka o RGB vrijednostima piksela učitane slike, pohranjivanje tih vrijednosti u obliku dvodimenzionalne matrice te provođenje FFT algoritma nad tom matricom. Zatim se na dobiveni rezultat primjenjuju različiti filtri koji omogućuju obradu originalne slike. Na kraju se provodi inverzna Fourierova transformacija kojom se dobiva originalna slika u filtriranom obliku.

Aplikacija je realizirana u obliku web stranice razvijene pomoću HTML-a, CSS-a i JavaScripta. Razvoj u obliku web stranice odabran je zbog mogućnosti lakšeg pokretanja na različitim računalima te zbog lakšeg prikaza svakog koraka od učitavanja originalne slike do dobivanja konačnog rezultata.

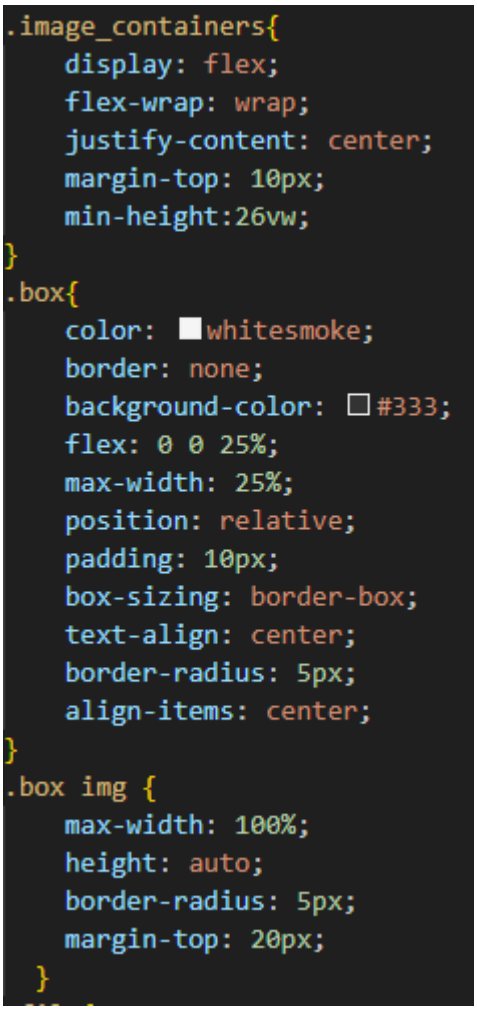
## 4.4. Opis izrade aplikacije za demonstraciju korištenja Fourierovih transformacija u obradi slika

Prvi korak pri izradi aplikacije bio je dizajn korisničkog sučelja. Odlučeno je da će se proces od učitavanja slike do konačnog rezultata prikazati u obliku četiri koraka od kojih će svaki prikazivati jednu sliku. Prvo se prikazuje originalna slika, zatim se prikazuje rezultat primjene FFT algoritma, nakon toga prikazuje se primijenjeni filtar i na kraju se prikazuje rezultat primjene inverznog FFT algoritma.

Navedene četiri slike i njihovi „spremnici“ prikazuju se u horizontalnoj liniji na sljedeći način:

```
<div class="image_containers">
    <div class="box">
        <div class="content">
            <label id="box_title">Početna slika</label>
            <img id="starting_image">
        </div>
    </div>
    <div class="box">
        <div class="content">
            <label id="box_title">FFT rezultat</label>
            <img id="fft_image">
        </div>
    </div>
    <div class="box">
        <div class="content">
            <label id="box_title">Rezultat filtriranja</label>
            <img id="reduction_result_image">
        </div>
    </div>
    <div class="box">
        <div class="content">
            <label id="box_title">Inverzni FFT rezultat</label>
            <img id="inverse_fft_image">
        </div>
    </div>
</div>
```

*Slika 37: HTML struktura spremnika za slike [snimka zaslona]*



```css
.image_containers{
    display: flex;
    flex-wrap: wrap;
    justify-content: center;
    margin-top: 10px;
    min-height:26vw;
}
.box{
    color: whitesmoke;
    border: none;
    background-color: #333;
    flex: 0 0 25%;
    max-width: 25%;
    position: relative;
    padding: 10px;
    box-sizing: border-box;
    text-align: center;
    border-radius: 5px;
    align-items: center;
}
.box img {
    max-width: 100%;
    height: auto;
    border-radius: 5px;
    margin-top: 20px;
  }
```

*Slika 38: CSS struktura spremnika za slike [snimka zaslona]*

Nakon izrade sučelja za prikaz svih koraka napisan je JavaScript kod za prikaz slika u crno-bijelom načinu rada (neovisno je li učitana slika u boji ili crno bijela). Prikaz u crno-bijelom načinu odabran je zato što omogućuje brže provođenje FFT algoritma.

Nakon implementacije učitavanje slike napisan je kod za izvlačenje podataka o pikselima slike (izvlačenje RGB vrijednosti svakog piksela slike) i spremanje tih piksela u dvodimenzionalnu matricu. Dimenzije polja odgovaraju rezoluciji slike (za svaki piksel sprema se samo jedna vrijednost jer u slučaju crno-bijelih slika RGB vrijednosti za svaki pojedini piksel su jednake).

Za provedbu FFT algoritma implementiran je JavaScript web worker. Web worker omogućuje pokretanje JavaScript koda u pozadini bez utjecaja na korisničko sučelje. Korištenje web workera odabrano je zbog blokiranja korisničkog sučelja pri provođenju FFT algoritma. Bez korištenja web workera pokretanje FFT algoritma

blokira cijelo korisničko sučelje sve dok se algoritam ne izvrši što stvara neugodno korisničko iskustvo, pogotovo pri učitavanju slika viših rezolucija.

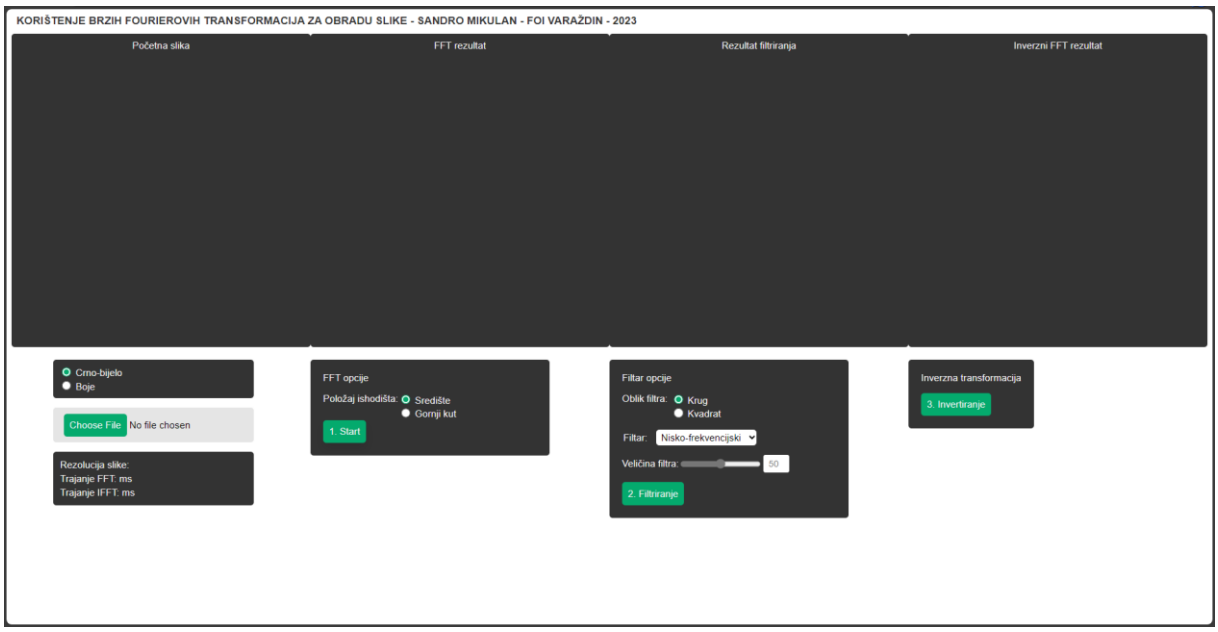
Zatim su implementirani filtri i način primjene filtera na rezultat FFT algoritma. Implementirano je ukupno sedam različitih filtera od kojih su tri za zamućivanje, tri za izvlačenje detalja i jedan za izoštravanje slike, filteri će kasnije biti detaljnije opisani.

Na kraju je implementiran kod za primjenu inverznog FFT algoritama, također unutar web workera, te je prikaz konačnog rezultata.

Uz sve navedeno također su implementirane funkcionalnosti za prebacivanje iz načina rada koji prikazuje slike u crno-bijeloj boji u način rada koji prikazuje slike u boji (samo u slučaju kada je originalna slika bila u boji). Implementiran je odabir filtera i raznih parametara za pojedini filtar te je implementiran odabir načina na koji će se provesti FFT algoritam (može se odabrati hoće li se visoke frekvencije spremati u središte 2D matrice/prikazivati u sredini slike ili u kutove 2D matrice/prikazivati u kutovima slike). Također je implementiran prikaz informacija o rezoluciji slike i trajanju FFT i inverznog FFT algoritma. Svi navedeni koraci izrade i implementirane funkcionalnosti bit će detaljnije objašnjeni u sljedećem poglavlju.

## 4.5. Opis funkcionalnosti aplikacije za demonstraciju korištenja Fourierovih transformacija nad slikama

Glavna funkcionalnost aplikacije jest primjena filtera za izoštravanje odnosno zamućivanje učitanih slika nad kojima je proveden FFT algoritam. Slika 39 prikazuje korisničko sučelje aplikacije:




*Slika 39: Korisničko sučelje aplikacije [snimka zaslona]*

Sučelje se može podijeliti u sedam cjelina od kojih svaka sadrži neke od funkcionalnosti:

- Kontrola boje učitane slike
- Učitavanje originalne slike
- Kontrola načina prikaza rezultata FFT algoritma
- Kontrola parametara filtara i odabir filtra
- Inverzna transformacija
- Prikaz podataka o originalnoj slici i trajanju FFT algoritma

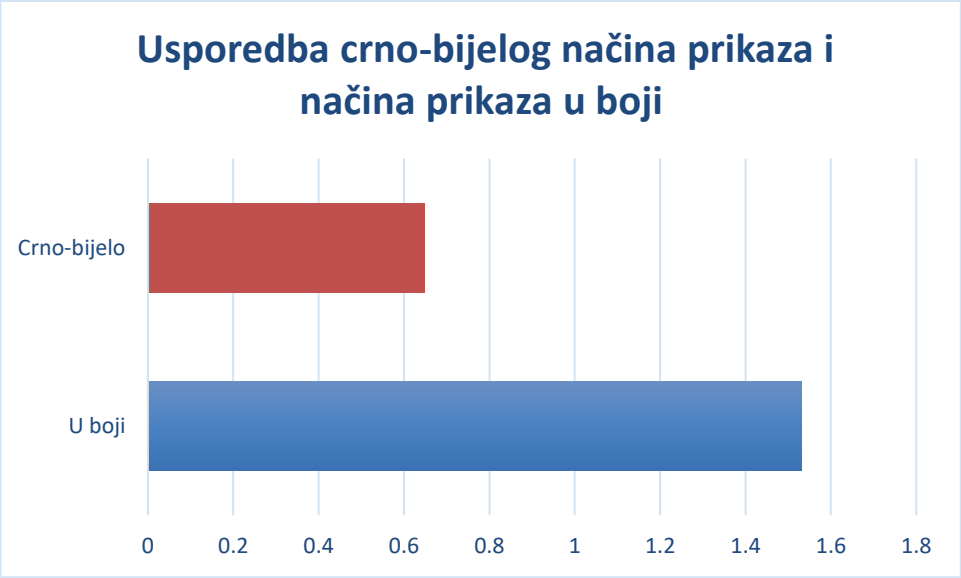
### Kontrola boje učitane slike

Prvi korak pri korištenju aplikacije jest odabrati hoće li se učitana slika prikazati i obrađivati kao crno-bijela ili u boji. Zadana postavka je prikaz i obrada crno-bijele slike jer osigurava brže izvođenje obrade. Na slici 40 prikazano je sučelje za kontrolu boje učitane slike:


*Slika 40: Sučelje za kontrolu boje učitane slike [snimka zaslona]*

Kada se odabere način prikaza u boji iz učitane slike izvlače se i spremaju sve RGB vrijednosti za sve piksele u slici, a kada je odabran crno-bijeli način prikaza prvo se obrađuju svi pikseli u slici tako da se uzme prosjek RGB vrijednosti svakog piksela te se dobivena vrijednost postavlja kao R, G i B vrijednost piksela od kojega je taj prosjek dobiven. Posljedica toga su  bolje performanse crno-bijelog načina prikaza od načina prikaza u boji, točnije obrada crno-bijelog načina prikaza je oko 3 puta brža od načina prikaza u boji. To je zato što se FFT algoritam provodi za svaku pojedinu RGB vrijednost tj. u načinu prikaza u boji FFT se mora provesti tri puta, dok se u crno-bijelom načinu prikaza provodi samo jednom te se dobivena vrijednost pri iscrtavanju slike kopira u preostale dvije komponente RGB vrijednosti.

Slika 41 prikazuje graf za usporedbu trajanja FFT algoritma u načinu prikaza u boji i crno-bijelom načinu prikaza za sliku rezolucije 512x512 piksela.




*Slika 41: Usporedba trajanja FFT algoritma u načinu prikaza u boji i crno-bijelom načinu prikaza [autorski rad]*

Test je proveden tri puta za svaki način prikaza te graf prikazuje prosjek tih testiranja, FFT algoritam u crno-bijelom načinu trajao je prosječno 0.647 sekunde dok je u  načinu u boji trajao prosječno 1.352 sekunde.

### Učitavanje originalne slike

Nakon odabira boje prikaza učitane slike prelazi se na učitavanje same slike, za učitavanje koristi se HTML *input* element te handleFileSelect() funkcija. Kod za HTML element i glavni dio funkcije prikazani su na slikama 42 I 43:

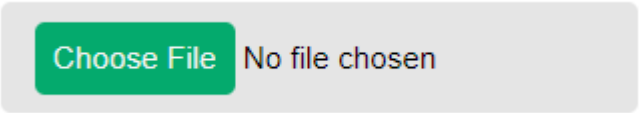
```
<input type="file" id="file"/>
```

*Slika 42: HTML struktura input elementa [snimka zaslona]*

```
function handleFileSelect(evt) {
    var files = evt.target.files;
    var selFile = files[0];
    var type=selFile.name.split('.').pop();
    var reader = new FileReader();
```

*Slika 43: Izgled glavnog djela handleFileSelect() funkcije [snimka zaslona]*

Sučelje elementa za odabir slike prikazano je na slici 44:




*Slika 44: Sučelje elementa za odabir slike [snimka zaslona]*

Pomoću gore prikazanog koda dobiva se i ekstenzija odabrane datoteke (u ovom slučaju odabiru se slike). Program podržava rad s nekoliko različitih formata slika (npr. .jpg/.jpeg, .bmp, .png). Nakon odabira željene slike stvara se prazni *image* element te se u njega sprema odabrana slika pomoću onLoad() funkcije unutar handleFileSelect() funkcije.

Dobivena slika se zatim iscrtava na pomoćnom *canvas* elementu koji je korisniku nevidljiv. Prikaz slike koja je vidljiva korisniku odrađen je pomoću HTML *image* elementa koji podatke o slici dobiva s pomoćnog *canvasa*. Isti pomoćni canvas element se koristi u svim koracima koji se provode u sklopu funkcionalnosti aplikacije.

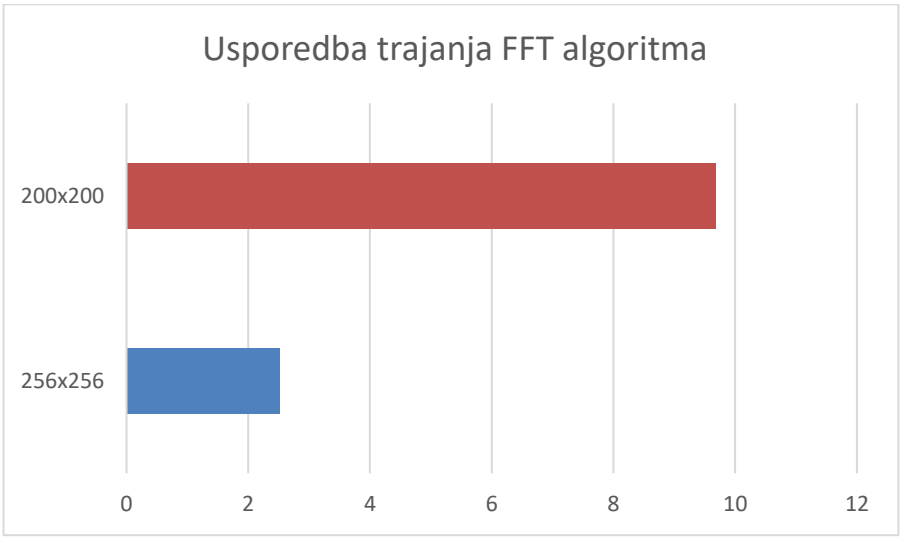
U ovom koraku se izvlače podaci o pikselima slike i spremaju u zasebnu 2D matricu, podaci se također ovisno o odabranoj opciji crno-bijelo ili boje obrađuju u ovom koraku.

Ono što najviše utječe na performanse aplikacije pri odabiru slike jest rezolucija odabrane slike, u pravilu što je rezolucija veća to više vremena treba za učitavanje same slike i za provedbu FFT algoritama. Vrijeme provedbe FFT algoritma direktno ovisi o broju piksela u slici pa je tako potrebno oko 4 puta više vremena za provedbu FFT algoritma nad slikom rezolucije 1024x1024 piksela nego što je potrebno za sliku rezolucije 512x512 piksela.

Uz rezoluciju slike veliku ulogu u brzini izvođenja ima činjenica je li rezolucija slike parni broj i ako je, da li je potencija broja 2. Zbog prirode FFT algoritama opisane u teorijskom dijelu rada najbrže rezultate daje rad sa slikama čija rezolucija je potencija broja 2 (npr. 128x128 ili 512x512). Ako je rezolucija parni broj, ali nije potencija broja 2, algoritam radi sporije, a ako je rezolucija neparni broj onda radi znatno sporije. Ova aplikacija ne radi sa slikama čija rezolucija ima neparne brojeve (npr. 127x127), razlog tome su izrazito spore performanse i problemi pri preslagivanju 2D polja dobivenog FFT algoritmom.

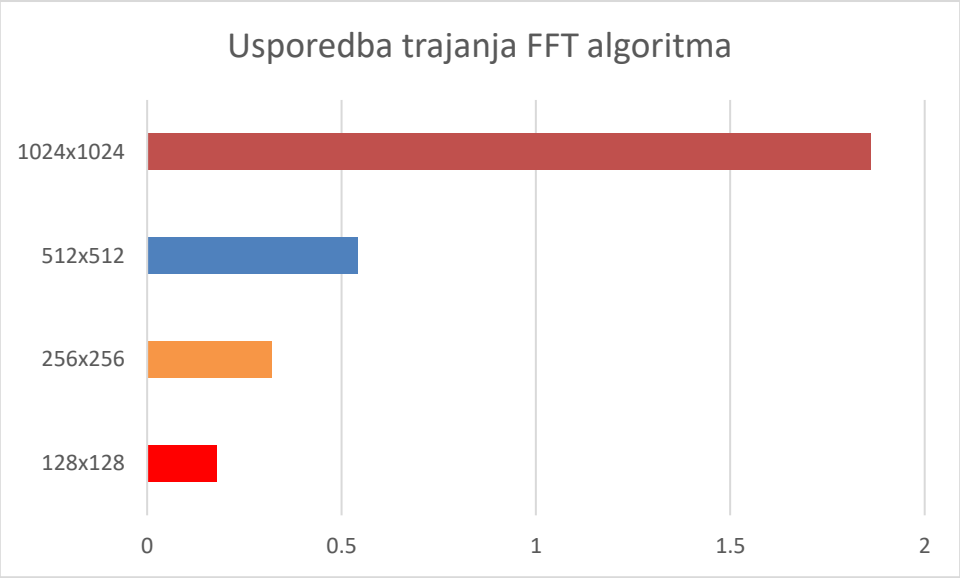
Na sljedećem grafu prikazana je usporedba vremena izvršenja FFT algoritma nad slikom s rezolucijom koja je potencija broja 2 (256x256) i nad slikom s rezolucijom koja je parni broj ali nije potencija broja 2 (200x200). Test je proveden tri puta za

svaku rezoluciju i na grafu je prikazan prosjek. Rezultati pokazuju da unatoč tome što slika rezolucije 200x200 ima manje piksela od slike rezolucije 256x256 potrebno je gotovo 4 puta duže da se provede FFT algoritam. Za ovaj test iskorištena je Mathjs biblioteka pa se vidi razlika u brzini u odnosu na Project Nayuki, no svrha ovog testa nije bila pokazati razliku u brzini već odnos brzina transformacije slika čije su dimenzije potencije broja 2 i slika čije su dimenzije parne ali nisu potencija broja 2.




*Slika 45: Usporedba trajanja FFT algoritma ovisno o rezoluciji [autorski rad]*

Na sljedećem grafu prikazana je razlika u vremenu izvođenja FFT algoritma za slike različitih rezolucija (128x128, 256x256, 512x512 i 1024x1024). Također su provedena tri testa za svaku sliku i graf prikazuje prosjek dobivenih vremena. Iz rezultata se vidi da se povećanjem broja piksela proporcionalno povećava vrijeme izvođenja. Tek se pri većim rezolucijama (usporedba 256x256 i 512x512) vidi očekivano 4 puta veće vrijeme izvođenja dok pri nižim rezolucijama razlika nije toliko primjetna.




*Slika 46: Usporedba trajanja FFT algoritma ovisno o broju piksela [autorski rad]*

**Kontrola načina prikaza rezultata FFT algoritma**

Prije provođenja FFT algoritma i primjene filtera potrebno je odabrati željene parametre. Sve kontrole i odabir parametara grupirani su u zasebne „kontejnere“. Sučelje za odabir načina prikaza rezultata FFT algoritma prikazano je na slici 47:

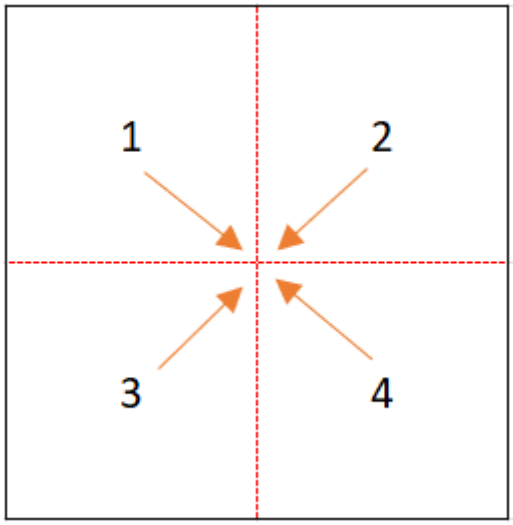

*Slika 47: Sučelje za odabir načina prikaza rezultata FFT algoritma [snimka zaslona]*

Glavne kontrole prikazane su u obliku tri gumba: Start, Filtriranje i Invertiranje. Start gumb se koristi za pokretanje FFT algoritma, gumb za filtriranje se koristi za primjenu odabranog filtera a gumb za invertiranje za pokretanje inverznog FFT algoritma.

Prije korištenja start gumba potrebno je odrediti željeni položaj ishodišta u rezultatima, ova opcija određuje način na koji će FFT rezultati biti prikazani i

spremljeni. Kod normalnog korištenja FFT funkcije unutar project Nayuki biblioteke podaci su spremljeni tako da se visoke frekvencije spremaju u kutove matrice a niske u središte (visoke frekvencije predstavljaju podatke o rubovima na učitanoj slici tj. mjestima gdje jedna boja oštro prelazi u drugu boju, niske frekvencije sadrže ostale podatke o slici).

Odabir položaja ishodišta u kutu podatke će spremiti i pokazati na gore opisani način. Odabir položaja ishodišta u središtu prebacuje podatke o visokim frekvencijama u središte a podatke o niskim frekvencijama u kutove. Invertiranje se postiže tako da se dobivena 2D matrica podjeli u 4 jednaka kvadranta te se kvadranti ispremještaju na sljedeći način:

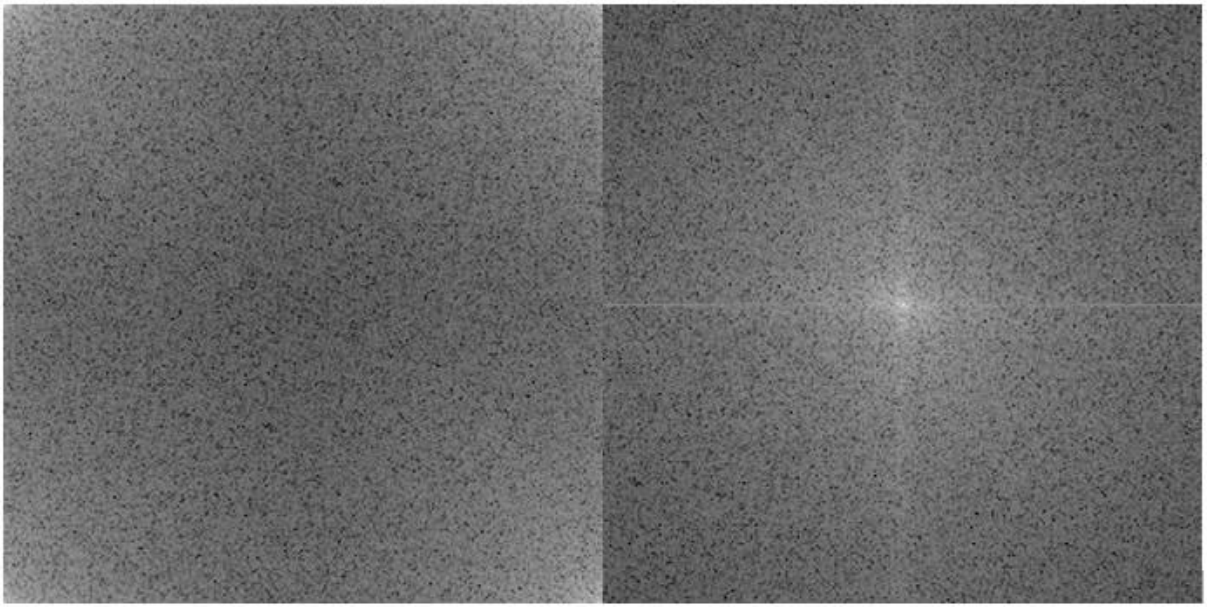



*Slika 48: Način invertiranja dobivene 2D matrice [autorski rad]*

U nastavku su prikazani kod koji provodi inverziju dobivenog 2d polja i konkretan rezultat koji se time postiže na primjeru slike rezolucije 256x256 piksela.

```
const numRows=array.length;
const numCols=array[0].length;
shiftedFFT = math.zeros([numRows, numCols]);
const halfNumRows = Math.floor(numRows / 2);
const halfNumCols = Math.floor(numCols / 2);
  for (let i = 0; i < numRows; i++) {
    for (let j = 0; j < numCols; j++) {
      const newRow = (i + halfNumRows) % numRows;
      const newCol = (j + halfNumCols) % numCols;
      shiftedFFT[newRow][newCol] = fftResultArray[i][j];
    }
  }
```

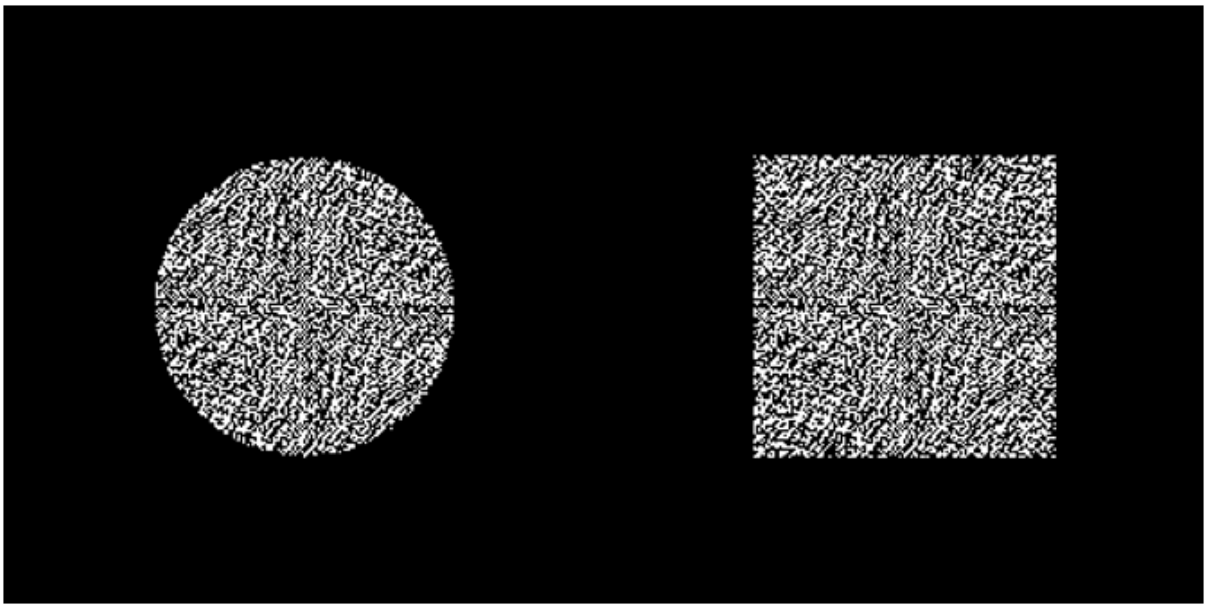
*Slika 49: Kod za invertiranje matrice [snimka zaslona]*



*Slika 50:Rezultat FFT algoritma s položajem ishodišta u kutu (lijevo) i položajem ishodišta u središtu (desno) [autorski rad]*

**Kontrola parametara filtara i odabir filtra**

Prije odabira željenog filtera potrebno je odrediti željeni oblik filtra, ovdje se određuje oblik odabranog visoko-frekvencijskog ili nisko-frekvencijskog filtera. Moguće je odabrati oblik kruga („circle“) ili pravokutnika („rectangle“). Na slici 51 prikazana je razlika između filtera u obliku kruga i pravokutnika za nisko-frekvencijski filtar s prikazom rezultata u središtu matrice:



*Slika 51: Razlika „krug“ i „pravokutnik“ oblika filtera [autorski rad]*

Na slici 52 prikazana je razlika u primjeni istog filtera ali u različitim oblicima, primjećuje se da je slika dobivena kružnim oblikom filtera nešto lošija, što je logično jer je veći broj frekvencija postavljen na 0 (krug zauzima manju površinu od pravokutnika).

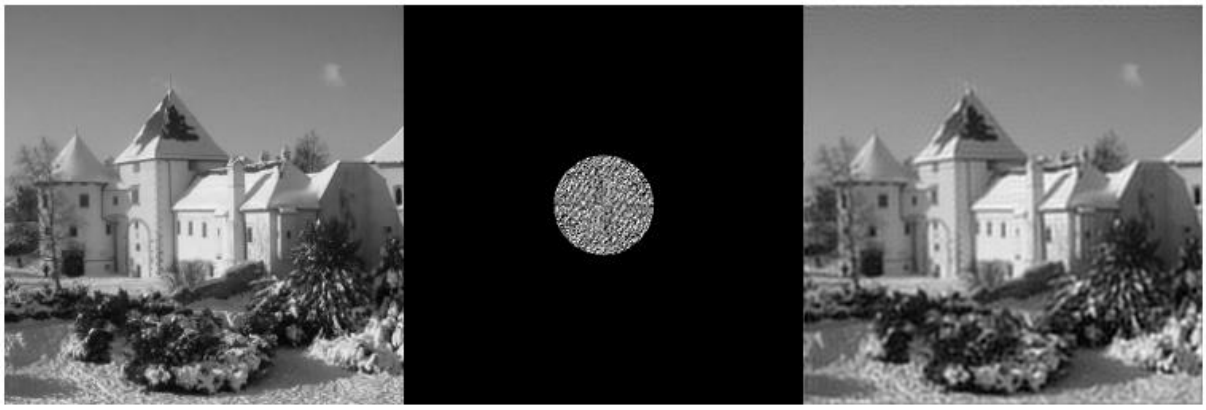
*Slika 52: Rezultat primjene istog filtera u različitim oblicima, krug (lijevo) i pravokutnik (desno) [autorski rad]*

Prije pritiska na gumb za filtriranje potrebno je odabrati željeni filtar. Implementirano je nekoliko različitih opcija za filtere za svaki pojedini način prikaza rezultata:

- Položaj ishodišta u središtu
    - Nisko-frekvencijski filtar
    - Visoko-frekvencijski filtar
    - Filtar za izoštravanje
    - Gaussov nisko-frekvencijski filtar
    - Gaussov visoko-frekvencijski filtar
- Položaj ishodišta u gornjem kutu
    - Nisko-frekvencijski filtar
    - Visoko-frekvencijski filtar

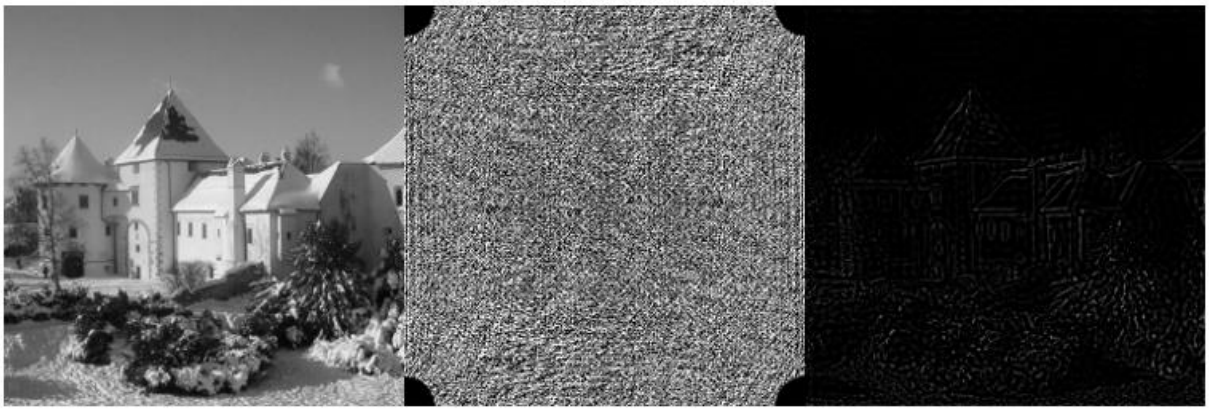
Nisko-frekvencijski filtar označava tip filtera koji ostavlja samo niske frekvencije, a vrijednosti visokih frekvencija postavlja na 0. Time se dobiva zamućena slika. Na slici

53 je primjer originalne slike, primijenjenog nisko-frekvencijskog filtera s položajem ishodišta u središtu i dobivenog rezultata:



*Slika 53: Prikaz rezultata primjene nisko-frekvencijskog filtera s položajem ishodišta u središtu [autorski rad]*

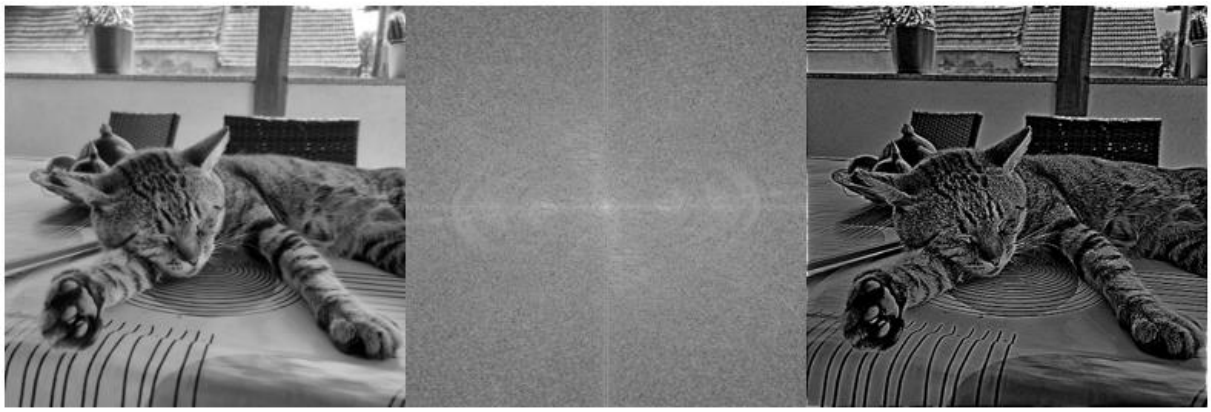
Visoko-frekvencijski filtar označava tip filtera koji ostavlja samo visoke frekvencije a vrijednosti niskih frekvencija postavlja na 0. Time se dobiva slika koja sadrži samo obrise originalne slike. Na slici 54 je primjer originalne slike, primijenjenog visoko-frekvencijskog filtera s položajem ishodišta u kutu i dobivenog rezultata:



*Slika 54:Prikaz rezultata primjene visoko-frekvencijskog filtera s položajem ishodišta u kutu [autorski rad]*

Filtar za izoštravanje omogućava izoštravanje mutnih/jako kompresiranih slika. Radi samo s FFT rezultatom koji ima niske frekvencije u središtu dobivene 2D matrice (u našem slučaju potrebno je odabrati opciju položaja ishodišta u središtu). Filtar radi tako da se s obzirom na udaljenost od središta dobivene 2D matrice vrijednosti frekvencija množe s određenim koeficijentom, udaljavanjem od središta koeficijent s kojim se množi se povećava, na taj se način pojačavaju visoke

frekvencije. Time se postiže bolja definicija obrisa slike i efekt izoštravanja. Na slici 55 je primjer originalne slike, primijenjenog filtra za izoštravanje i dobivenog rezultata:



*Slika 55: Prikaz rezultata primjene filtera za izoštravanje [autorski rad]*

Primjenom nisko-frekvencijskog filtera pogotovo kada se želi postići veća razina zamućivanja na slici koja se dobije kao rezultat pojavljuju se artefakti, tj. iz svakog oštrog ruba na originalnoj slici počinju se širiti obrisi tog ruba. To se događa zbog načina primjene nisko-frekvencijskog filtera. Do sad pokazani filteri imaju oštru granicu nakon koje se frekvencije postavljaju na 0, i taj oštri prijelaz uzrokuje pojavu artefakata. Jedan od načina na koji se taj problem rješava jest postepeni prijelaz iz frekvencija čije se vrijednosti ne diraju do frekvencija čije se vrijednosti postavljaju na 0. Postoji nekoliko različitih filtera koji postižu željeni postepeni prijelaz te su oni objašnjeni u teorijskom dijelu rada. Za potrebe praktičnog dijela korišten je Gaussov nisko-frekvencijski filtar, na slikama 56 i 57 prikazan je kod za implementaciju tog filtera i vizualizacija primjene filtera na rezultat Fourierove transformacije.

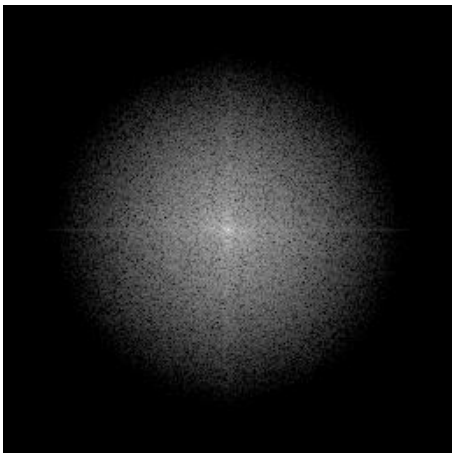
```
function gausslowPass(result,ctx_helper,slider_value,mode){
    if(mode=="greyscale_mode"){
        const rows = result.length;
        const cols = result[0].length;
        const middleRow = Math.floor(rows / 2);
        const middleCol = Math.floor(cols / 2);
        var sigma=slider_value;

        for (let i = 0; i < rows; i++) {
            for (let j = 0; j < cols; j++) {
                const distance=Math.sqrt(Math.pow(i-middleRow,2)+Math.pow(j-middleCol,2));
                const filterValue = Math.exp(-(distance ** 2) / (2 * sigma ** 2));
                result[i][j][0] *= filterValue;
                result[i][j][1] *= filterValue;
                ctx_helper.fillStyle = `rgb(${Math.log(Math.abs(result[i][j][0]))*15},${Math.log(Math.abs(result[i][j][0]))*15},${Math.log(Math.abs(result[i][j][0]))*15})`;
                ctx_helper.fillRect(i, j, 1, 1);

            }
        }

        return result;
    }
```

*Slika 56: Kod za implementaciju Gaussovog nisko-frekvencijskog filtera [snimka zaslona]*



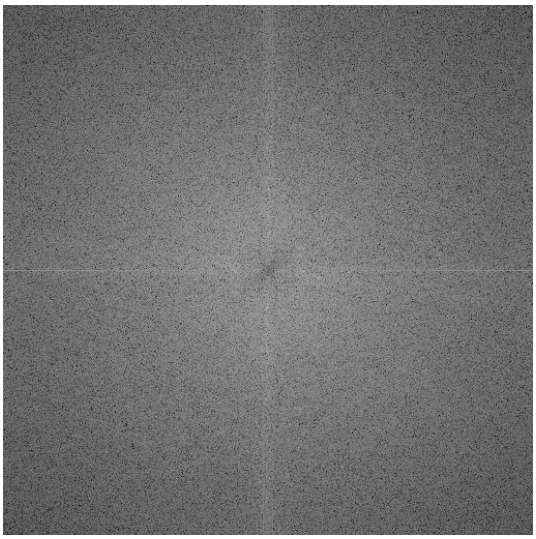
*Slika 57: Vizualizacija primjene Gaussovog nisko-frekvencijskog filtera [autorski rad]*

Na slikama 58 i 59 prikazana je usporedba primjene običnog nisko-frekvencijskog filtera (lijevo) i Gaussovog nisko-frekvencijskog (desno) u crno-bijelom načinu prikaza i načinu prikaza u boji. Vidi se da se primjenom Gaussovog nisko-frekvencijskog filtera rješava problem artefakata.

*Slika 58: Usporedba primjene običnog nisko-frekvencijskog filtera i Gaussovog nisko-frekvencijskog filtera (crno-bijeli način prikaza) [autorski rad]*

*Slika 59: Usporedba primjene običnog nisko-frekvencijskog filtera i Gaussovog nisko-frekvencijskog filtera (način prikaza u boji) [autorski rad]*

Isti problem javlja se kod primjene visoko-frekvencijskog filtera, na rezultatu se pojavljuju artefakti. Stoga je konstruiran Gaussov visoko-frekvencijski filtar. Filtar radi na istom principu kao i Gaussov nisko-frekvencijski s par izmjena. Kreira se kopija originalnih rezultata te se na nju primjenjuju vrijednosti filtera. Zatim se izračunate filtrirane vrijednosti oduzimaju od originalnih vrijednosti čime se postiže efekt visoko-frekvencijskog filtera. Na slikama 60 i 61 prikazan je primijenjeni visoko-frekvencijski filtar na rezultate transformacije i razlika primjene normalnog visoko-frekvencijskog filtera i Gaussovog visoko-frekvencijskog filtera. Iz rezultata se vidi da se postiže željeni efekt bez pojave artefakata.



*Slika 60: Prikaz primjene Gaussovog visoko-frekvencijskog filtera*

*Slika 61: Razlika običnog visoko-frekvencijskog filtera i Gaussovog visoko-frekvencijskog filtera*

Zadnja opcija koja se nudi je slider „Veličina filtra“ kojim se određuje veličina nisko-frekvencijskog ili visoko-frekvencijskog filtera u kružnom ili pravokutnom obliku filtera. Povećanjem/smanjenjem vrijednosti slidera povećava/smanjuje se udio frekvencija koje se postavljaju na 0 ovisno o odabranom filtru. Na slikama 62 i 63 prikazani su rezultati nisko-frekvencijskog filtera s ishodištem u kutu postavljenog na 50% odnosno 75% i rezultati dobiveni primjenom tih filtera (slika je rezolucije 256x256):

*Slika 62: Rezultat primjene nisko-frekvencijskog filtera postavljenog na 50% (snimka zaslona) [autorski rad]*

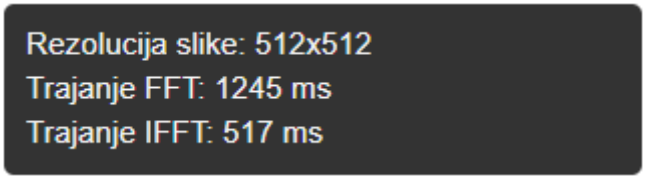
*Slika 63: Rezultat primjene nisko-frekvencijskog filtera postavljenog na 75% (snimka zaslona) [autorski rad]*

### Inverzna transformacija

Klikom na gumb za invertiranje pokreče se inverzni FFT algoritam nad 2D matricom na koju je primijenjen određeni filter, primjenom inverznog FFT algoritma dobiva se originalna slika u zamućenom/izoštrenom obliku.

### Prikaz podataka o originalnoj slici i trajanju FFT algoritma

Ovaj dio sučelja prikazuje informacije o rezoluciji odabrane slike te vremena izmjerena kod izvođenja FFT algoritma i inverznog FFT algoritma prikazana u milisekundama. Na slici 66 prikazan je izgled sučelja za prikaz podataka:

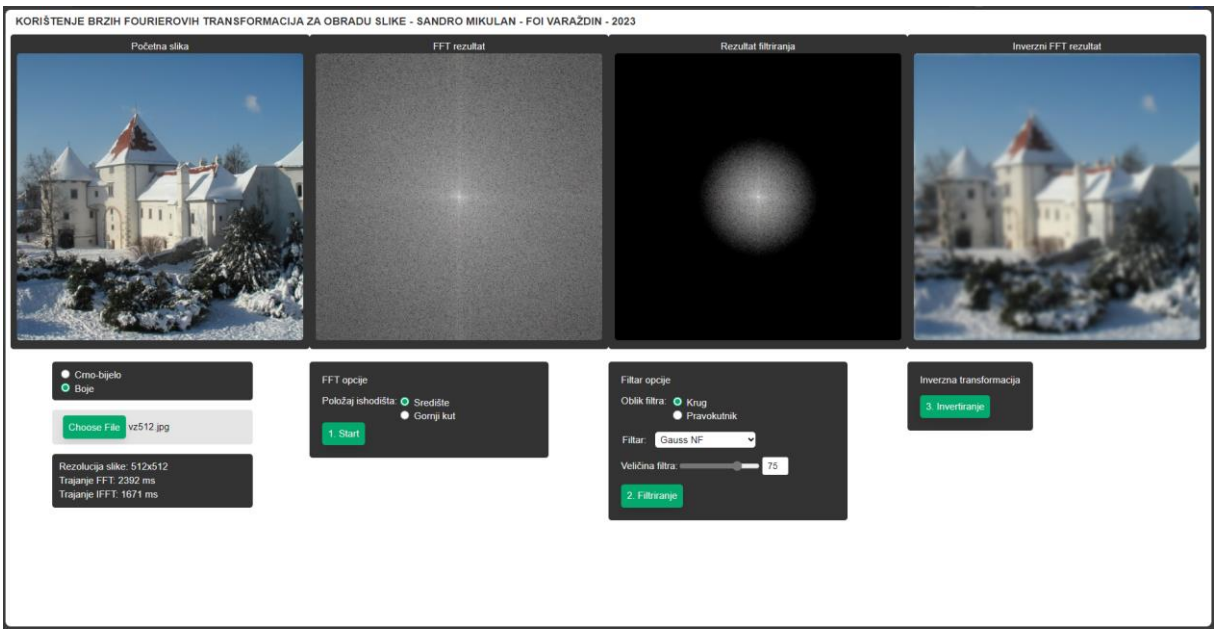



*Slika 64: Izgled sučelja za prikaz podataka [snimka zaslona]*

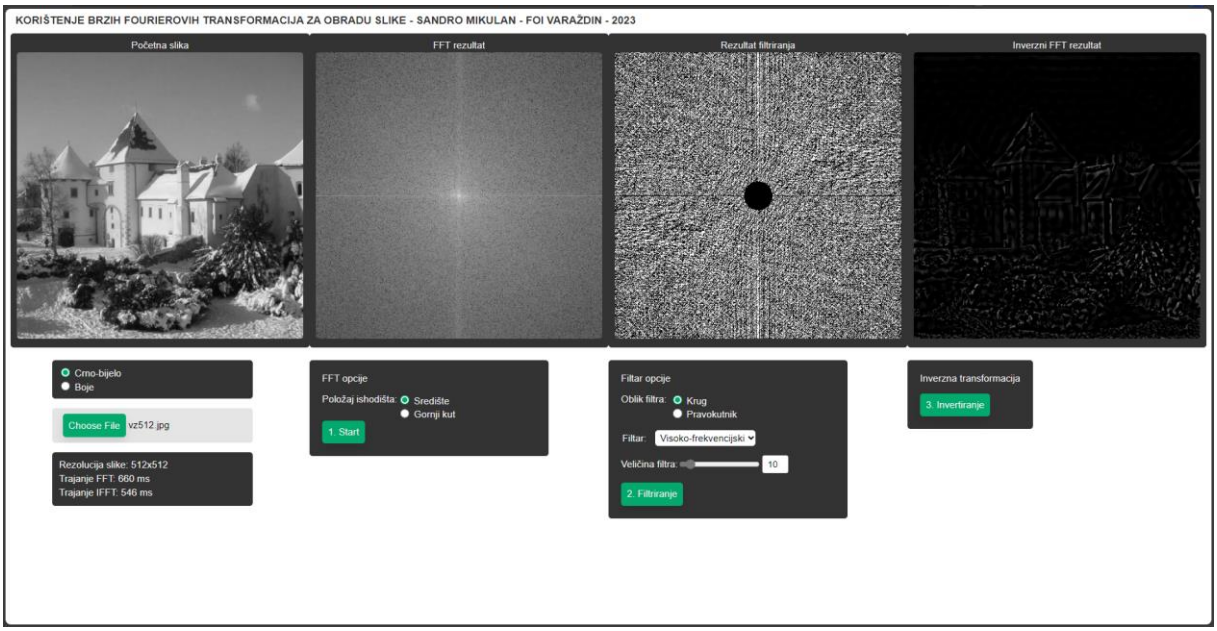
### Prikaz cjelokupne funkcionalnosti aplikacije

Prikazuju se četiri zasebna koraka u obliku slika, prvo je prikazana originalna slika, zatim rezultat primjene FFT algoritma, zatim rezultat primjene željenog filtera i na kraju rezultat primjene inverznog FFT algoritma

Na slikama 64 i 65 prikazan je izgled cjelokupnog sučelja s provedenim svim koracima u načinu prikaza u boji i crno bijelom načinu prikaza:




*Slika 65: Prikaz provedenih svih koraka u načinu prikaza u boji [snimka zaslona]*




*Slika 66: Prikaz provedenih svih koraka u crno-bijelom načinu prikaza [snimka zaslona]*

# 5. Zaključak

Ovaj rad pruža uvid u mnoge kompleksne teme iz područja matematike i računalne grafike. Na prvi pogled te teme su teške za razumjeti i objasniti, posebno kada se gleda matematička pozadina. Stoga je jedan od ciljeva ovoga rada bio pružiti razumljiv pregled Fourierovih transformacija i njihovih primjena bez potrebe za velikim predznanjem.

U prvom dijelu rada dan je uvod u ulogu i važnost Fourierovih transformacija kroz opis povijesti razvoja i primjena Fourierovih transformacija. Zatim je kroz primjere prikazano funkcioniranje Fourierovih transformacija, počevši od Fourierovog reda koji se može smatrati posebnim slučajem Fourierove transformacije.

Uvod u matematičku pozadinu poslužio je kao uvod u praktične slučajeve korištenja kroz opise diskretnih Fourierovih transformacija i algoritma brzih Fourierovih transformacija. Na kraju je dan pregled slučajeva korištenja u računalnoj grafici na temelju kojeg se može zaključiti da FFT algoritam ima široku primjenu te da bez njega mnoge moderne tehnologije ne bi postojale.

Kao praktični primjer uzet je slučaj korištenja za obradu i filtriranje slika. Razvijena je web aplikacija koja omogućava učitavanje slika u različitim formatima, pretvaranje slike u frekvencijsku domenu pomoću algoritma brzih Fourierovih transformacija, primjenu raznih filtera na slike te vraćanje modificirane originalne slike pomoću inverznih transformacija. Razvijena aplikacija sa svim svojim funkcionalnostima pruža jedinstven način prikaza primjene brzih Fourierovih transformacija. Naime na internetu postoje mnoge web stranice koje nude dio funkcionalnosti koje su razvijene za potrebe ovog praktičnog rada, ali nisam našao ni jednu koja bi nudila ovako zaokružen i jednostavan pristup pogodan za demonstraciju i korištenje u nastavi.

Kada se općenito gleda povijest, sadašnjost i budućnost smatram da će Fourierove transformacije ostati jednako važne kao što su sada i da ovaj rad dokazuje tezu postavljenu u uvodu tj. da je FFT algoritam jedan od najvažnijih algoritama u povijesti, jer se krije u pozadini gotovo svakog procesa obrade signala što predstavlja temelj gotovo svih tehnologija za prikaz vizualnih i zvučnih sadržaja.

# 6. Popis literature


[1] Margaret Rouse, „Fourier Transform“, 2020. Dostupno: What is the Fourier Transform? - Definition from Techopedia [pristupano 4.9.2023].

[2] J. Dongarra, F. Sullivan, „Guest Editors Introduction to the top 10 algorithms“, 2000. Dostupno: c1022.pdf (ieeecomputer.org) pristupano [8.9.2023].

[3] Desmos graphing calculator, dostupno: Desmos | Graphing Calculator, pristupano [8.9.2023].

[4] SageMath, the Sage Mathematics Software System (Version 9.5), The Sage Developers, 2023, https://www.sagemath.org.

[5] Mathjs, dostupno: math.js | an extensive math library for JavaScript and Node.js (mathjs.org), pristupano [8.9.2023].

[6] Project Nayuki, „Free small FFT in multiple languages“, 2021. Dostupno: Free small FFT in multiple languages (nayuki.io) pristupano [8.9.2023].

[7] The Fourier Transform.com, „The Fourier Trnasform.com“, 2010. Dostupno: Fourier Transform (thefouriertransform.com) [pristupano 4.9.2023].

[8] Veritasium, „The Remarkable Story Behind The Most Important Algorithm Of All Time“, 2022. Dostupno: (16) The Remarkable Story Behind The Most Important Algorithm Of All Time - YouTube [pristupano 4.9.2023].

[9] Weisstein, Eric W, „Fourier Series“, 2023. Dostupno: Fourier Series -- from Wolfram MathWorld [pristupano 4.9.2023].

[10] Saša Krešić Jurić, „Parcijalne diferencijalne jednadžbe Skripta-radna verzija“, bez.dat. Dostupno: pdj_skripta_v4.pdf (unist.hr) pristupano [11.9.2023].

[11] Murray R. Spiegel, „Schaum's outline of theory and problems of Fourier analysis with applications to boundary value problems“, 1974. Dostupno: fourier_series_1.pdf (ksu.edu.sa) pristupano [11.9.2023].

[12] Dr. Trefor Bazett, „Intro to FOURIER SERIES: The Big Idea“, 2021. Dostupno: (16) Intro to FOURIER SERIES: The Big Idea - YouTube pristupano [4.9.2023].

[13] Weisstein, Eric W, „Fourier Series—Square Wave“, 2023. Dostupno: Fourier Series--Square Wave -- from Wolfram MathWorld pristupano [4.9.2023].

[14] Better Explained, „An Interactive Guide To The Fourier Transform“, bez.dat. Dostupno: An Interactive Guide To The Fourier Transform – BetterExplained pristupano [4.9.2023].

[15] NTI Audio, „Fast Fourier Transformation FFT – Basics“, bez.dat. Dostupno: FFT (nti-audio.com) pristupano [4.9.2023].
[16] Erik Cheever, „Introduction to the Fourier Transform“, 2005. Dostupno: Introduction to the Fourier Transform (swarthmore.edu) pristupano [4.9.2023].
[17] Kolegij računalne grafike, „Fourierove transformacije“, bez.dat. PDF dokument.
[18] Lain Explains Signals, Systems, and Digital Comms, „Fourier Transform Equation Explained“, 2019. Dostupno: (16) Fourier Transform Equation Explained - YouTube pristupano [4.9.2023].
[19] Qingkai Kong, Timmy Siauw, Alexandre Bayen, „Python programming and numerical methods“, 2020. Dostupno: Discrete Fourier Transform (DFT) — Python Numerical Methods (berkeley.edu) pristupano [4.9.2023].
[20] Simon Xu, „Discrete Fourier Transform – Simple Step by Step“, 2015. Dostupno: (16) Discrete Fourier Transform - Simple Step by Step - YouTube pristupano [4.9.2023].
[21] Coursera, „What Is Computer Graphics? A Guide to Getting Started“, 2023. Dostupno: What Is Computer Graphics? A Guide to Getting Started | Coursera pristupano [4.9.2023].
[22] sakshiparkh23, „Introduction to Computer Graphics“, 2023. Dostupno: Introduction to Computer Graphics - GeeksforGeeks pristupano [4.9.2023].
[25] Stephen Gruppetta, „How to Create Any Image Using Only Sine Functions | 2D Fourier Transform in Python“, 2021. Dostupno: 2D Fourier transform in Python: Create any image using only sine functions (thepythoncodingbook.com) pristupano [4.9.2023].
[26] A. Zisserman, „Lecture 2: 2D Fourier transforms and applications“, 2014. Dostupno: Microsoft PowerPoint - lect2.pptx (ox.ac.uk) pristupano [11.9.2023]
[27] University of Verona , „2D Discrete Fourier Transform (DFT), 2010. Dostupno: Microsoft PowerPoint - IP-L3(2D-DFT)-2010.ppt (univr.it) pristupano [4.9.2023].
[28] The University of New Mexico, „Introduction to fourier transforms for image processing“, bez.dat. Dostupno: Introduction to the Fourier Transform (unm.edu) pristupano [4.9.2023].
[29] University of Verona, „Image Enhancement in the frequenvy domain“, bez.dat. Dostupno: IP-L6-GW-Chap4-Enhancement-F-domain.pptx (univr.it) pristupano [4.9.2023].

[30] Wikipedia, „Edge detection“, 2023. Dostupno: Edge detection - Wikipedia pristupano [4.9.2023].
[31] Xuewen Chen, „FFT Based Procedure for Texture Synthesis“, bez.dat. Dostupno: FFT Based Procedure for Texture Synthesis (cmu.edu) pristupano [4.9.2023]
[32] A. Krista Bird, B. Thomas Dickerson, C. Jessica George, „Techniques for Fractal Terrain Generation“, 2013. Dostupno: Techniques for Fractal Terrain Generation (williams.edu) pristupano [4.9.2023]

**Slike**

[23] Urban sign & print, “Vector vs Raster Graphics: What is the Differerence?“, 2019. Dostupno: Vector vs Raster Graphics: What is the Difference? | urbansignandprint.com pristupano [4.9.2023]
[24] The Printing Connection, „Raster Images vs. Vector Graphics“, bez.dat. Dostupno: Raster Images vs. Vector Graphics | The Printing Connection (printcnx.com) pristupano [4.9.2023]

# 7. Popis slika

# 8. Popis tablica