Ispitna dokumentacija projekta

# Analiza problema

Problematika ovog projekta jeste detekcija ivica unutar slike. Urađena je na 2 različita načina. Prvi je korišćenje Prewitt operatora koji filtrira sliku operatorima za horizontalne i vertikalne ivice i pomoću njih određuje da li se u nekoj tački nalazi ivica ili ne. Drugi način vrši detekciju ivica proveravajući okolinu tačke. Za oba načina, implementirana je serijska i paralelna verzija.

# Koncept rešenja

Serijsko rešenje prewitt operatora izvedeno je prolaskom kroz svaki piksel slike i primenjivanjem prewitt filtera na svaki piksel.

Prewitt filter prolazi kroz okolinu nekog piksela slike, a veličina te okoline zavisi od dimenzija filtera. Množe se piksel iz polja okoline sa odgovarajućim poljem filter matrice. To se vrši i za horizontalni i za vertikalni filter, a vrednosti koje se dobijaju se sabiraju za horizontalni i vertikalni filter ponaosob. Zatim se apsolutne vrednosti ta 2 zbira sabiraju, i ako su ispod granice od 128 pretvaraju se u 0 (belu boju) ili ako su iznad u 255 (crnu boju).

Paralelna verzija prewitt operatora pravi grupu od 4 zadatka gde svaki uzima po četvrtinu matrice koja predstavlja ulaznu sliku. Oni se tako dele dok se ne dostigne širina koja je manja ili jednaka šesnaestini pune širine slike. Tada se prelazi u rešenje koje je jednako serijskom, samo za podmatricu koja je pripala ovom zadatku.

Serijsko rešenje 2. edge detection algoritma takođe prolazi kroz svaki piksel slike i primenjuje filter za detektovanje ivica.

Filter prvo postavlja početne vrednosti za promenljive P (0) i O (1). Zatim prolazi kroz okolinu piksela, gde veličina okoline zavisi od toga kolika udaljenost je odabrana. Za svaku tačku u okolini (bez samog piksela koji se filtrira) se proverava da li je njegova vrednost veća od granice od 128. Ako jeste, vrednost P se postavlja na 1, inače se vrednost O postavlja na 0. Kada se prođe kroz celu okolinu, apsolutna vrednost od O se oduzima od apsolutne vrednosti od P, i ta vrednost se transformiše u opseg 0-255 koji predstavlja spektar od bele do crne boje. Drugim rečima, ako se dobije vrednost 0, i ostaće 0, ako se dobije 1, pretvoriće se u 255.

Paralelno rešenje kreira zadatke na isti način kao i paralelno rešenje prewitt operatora, s tim da kada širina dobijene podmatrice zadatka postane manja ili jednaka šesnaestini širine slike, prelazi se na rešenje jednako serijskom za 2. edge detection algoritam.

Bitno je napomenuti da će izlazne slike uvek biti praktično malo manje nego ulazne. To se dešava zbog načina na koji oba načina detekcija ivice funkcionišu. I jedan i drugi način za svaki piksel analiziraju neku njenu okolinu. To je nemoguće uraditi na ivičnim pikselima koji nemaju dovoljno susednih piksela za ovakvu analizu. Zato će ti ivični pikseli biti ignorisani i činiti samo crnu granicu oko izlazne slike.

# Programsko rešenje

Najbitnije kreirane funkcije:

void filter\_serial\_prewitt(int\* inBuffer, int\* outBuffer, int width, int height)

Izvršava serijsku verziju prewitt operator rešenja problema. Ima ugnježden for loop koji prolazi kroz sliku i poziva apply\_prewitt\_filter. inBuffer – bafer ulazne slike, outBuffer – bafer izlazne slike, width – širina slike, height – visina slike.

int apply\_prewitt\_filter(int\* inBuffer, int row, int column, int width)

Primenjuje prewitt filter na uneti piksel slike. inBuffer – bafer ulazne slike, row – red inBuffer polja koji će biti filtriran, column – kolona inBuffer polja koje će biti filtrirano, width – širina slike.

void perform\_parallel\_prewitt(int\* inBuffer, int\* outBuffer, int width, int height, int row, int column, int fullWidth)

Izvršava paralelnu verziju prewitt operator rešenja problema. Pravi 4 zadatka gde svaki pokriva po četvrtinu matrice, a kada se širina podmatrice smanji na manje ili jednako od šesnaestine širine slike izvršava se serijsko rešenje prewitt problema. inBuffer – bafer ulazne slike, outBuffer – bafer izlazne slike, width – širina dela inBuffer matrice za koju je ovaj zadatak odgovoran, height – visina dela matrice za koju je ovaj zadatak odgovoran, row – početni red inBuffer matrice za koju je zadatak odgovoran, column – početna kolona inBuffer matrice za koju je zadatak odgovoran, fullWidth – širina slike.

void filter\_serial\_edge\_detection(int\* inBuffer, int\* outBuffer, int width, int height)

Izvršava serijsku verziju 2. algoritma za detekciju ivica. U ugnježdenom for loopu poziva se funkcija apply\_edge\_filter. inBuffer – bafer ulazne slike, outBuffer – bafer izlazne slike, width – širina slike, height – visina slike.

int apply\_edge\_filter(int\* inBuffer, int row, int column, int width)

Primenjuje filter 2. algoritma na jedno polje matrice koje predstavlja sliku. inBuffer – bafer ulazne slike, row – red inBuffer polja koji će biti filtriran, column – kolona inBuffer polja koje će biti filtrirano, width – širina slike.

void perform\_parallel\_edge\_detection(int\* inBuffer, int\* outBuffer, int width, int height, int row, int column, int fullWidth)

Izvršava paralelnu verziju 2. algoritma za rešenje problema. Pravi 4 zadatka gde svaki pokriva po četvrtinu matrice, a kada se širina podmatrice smanji na manje ili jednako od šesnaestine širine slike izvršava se serijska implementacija 2. Algoritma. inBuffer – bafer ulazne slike, outBuffer – bafer izlazne slike, width – širina dela inBuffer matrice za koju je ovaj zadatak odgovoran, height – visina dela matrice za koju je ovaj zadatak odgovoran, row – početni red inBuffer matrice za koju je zadatak odgovoran, column – početna kolona inBuffer matrice za koju je zadatak odgovoran, fullWidth – širina slike.

Ove funkcije su implementirane tako da odgovaraju konceptu rešenja iznad.

Korišćena je i već napravljena klasa BitmapRawConverter koja koristi funkcije i klase iz EasyBMP biblioteke. Neke od korišćenih funkcija te klase su getWidth i getHeight pozvane nad BitmapRawConverter inputFile. Koriste se da bi dobili širinu i visinu ulazne slike, u pikselima.

# Ispitivanje

Karakteristike računara na kome je rađen projekat kao i ispitivanje:

Procesor: Intel Core i5-4460 3.20 GHz 4 core 4 threads

RAM: 16 GB

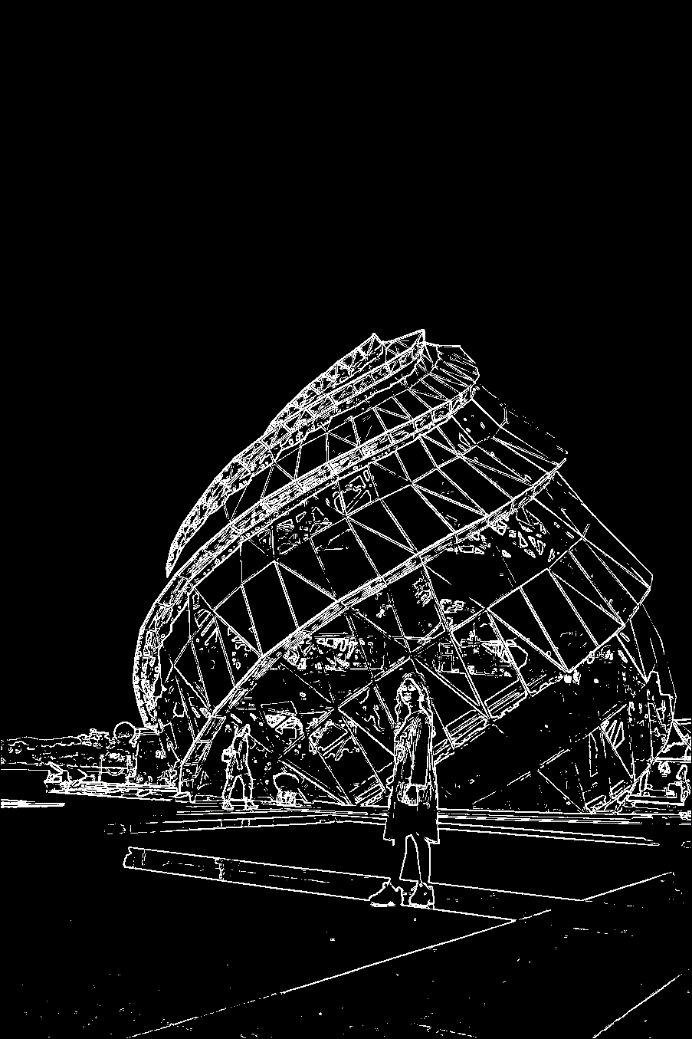
Operativni sistem: Windows 10, 64-bitni OS

Slika: staklena zgrada (903x1355)



Prewitt 3x3

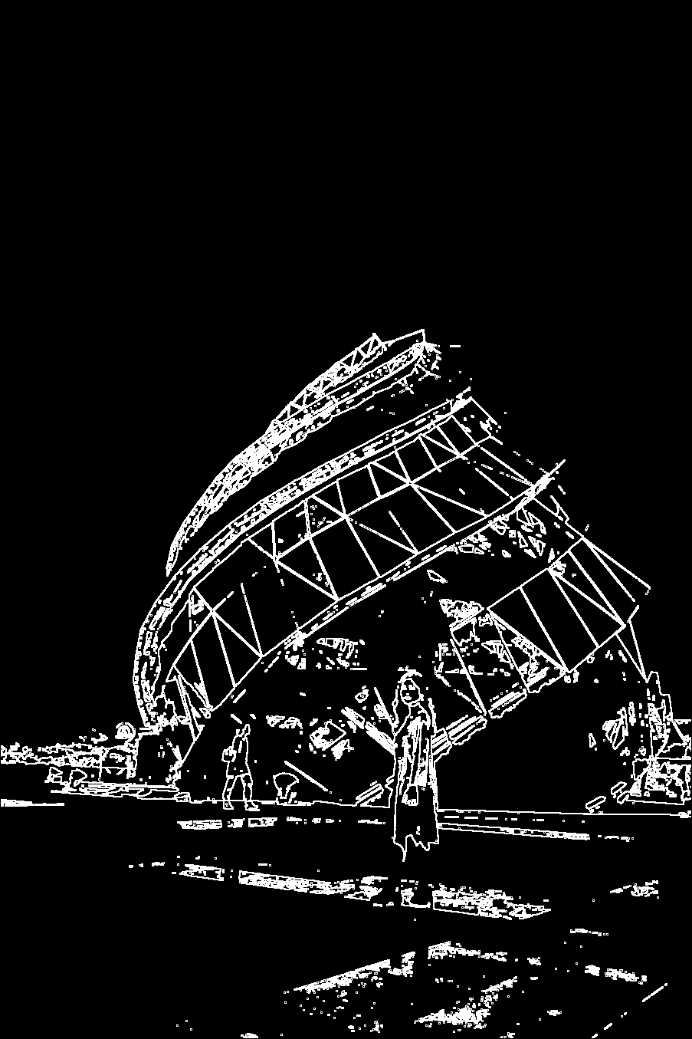
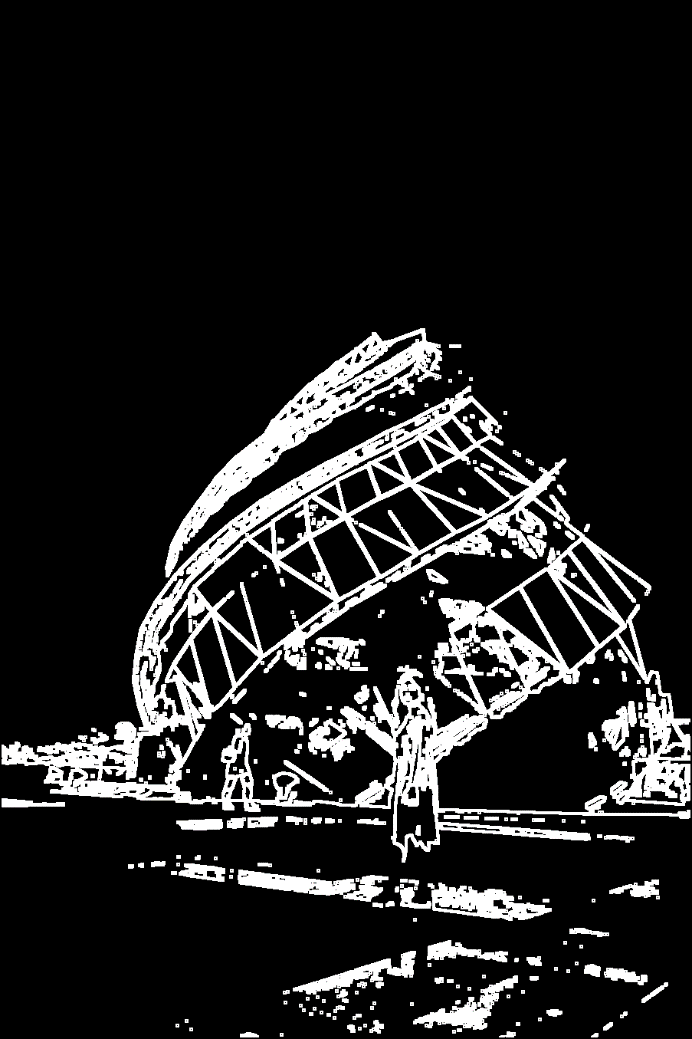
Prewitt 5x5



Udaljenost=1



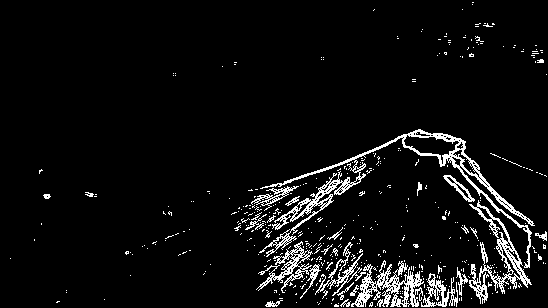
Udaljenost=2

Slika: planina (548x308)



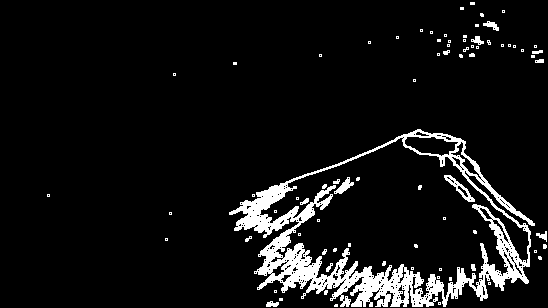
Prewitt 3x3:



Prewitt 5x5:



Edge udaljenost=1:



Edge udaljenost=2:



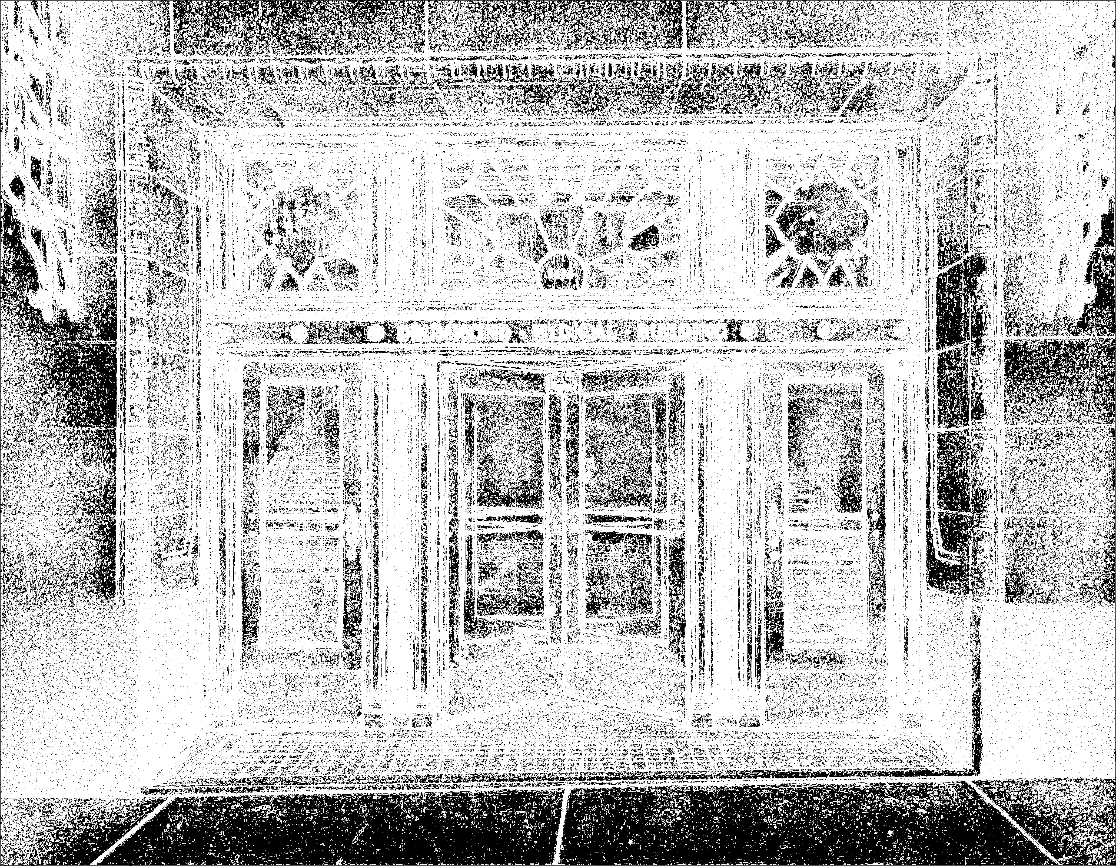
Slika: ulaz (4128x3204)



Prewitt 3x3:



Prewitt 5x5:



Edge udaljenost=1:



Edge udaljenost=2:



Ispod se nalazi tabela formirana nakon ispitivanja. Uzeta vremena su prosečne vrednosti dobijene nakon 5 pokretanja programa.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Naziv slike | Dimenzije slike | Dim. Prewitt filtera | Vreme Prewitt (serijski) | Vreme Prewitt (paralelni) | Ubrzanje za Prewitt | Udaljenost za 2. način | Vreme 2. način (serijski) | Vreme 2. način (paralelni) | Ubrzanje za 2. način |
| Planina | 548x308 | 3x3 | 16,2ms | 8ms | 2,02x | 1 | 10ms | 3,4ms | 2,94x |
| Planina | 548x308 | 5x5 | 31ms | 12,4ms | 2,5x | 2 | 17,4ms | 5,2ms | 3,35x |
| Staklena zgrada | 903x1355 | 3x3 | 125,4ms | 41,6ms | 3,01x | 1 | 88,2ms | 24,8ms | 3,56x |
| Staklena zgrada | 903x1355 | 5x5 | 235ms | 60ms | 3,92x | 2 | 139,4ms | 42,2ms | 3,3x |
| Ulaz | 4128x3204 | 3x3 | 1249,4ms | 370,8ms | 3,37x | 1 | 925,2ms | 241ms | 3,84x |
| Ulaz | 4128x3204 | 5x5 | 2304ms | 686,4ms | 3,36x | 2 | 1513,6ms | 425,8ms | 3,56x |

Ispod i na sledećoj strani se nalaze grafici koji pokazuju ubrzanje spram povećanja dimenzija problema.

# Analiza rezultata

Ovaj projekat je predstavio zanimljiv problem koji je bio pogodan za optimizaciju. Verujem da je postignuto očekivano ubrzanje. Sa 3x3 prewitt filterom ili udaljenost=1 2. algoritmom vidimo linearno ubrzanje sa povećanjem dimenzija slike. Na početku sa prewitt filterom imamo 2x ubrzanje, što je naravno korisno ali ni blizu dovoljno dobro sa obzirom na to da računar ima 4 jezgra. Ovo se dešava zbog niske dimenzije problema, gde nema dovoljno posla da bi paralelizovanje napravilo veću razliku. Posao se završi previše brzo i sa serijskim rešenjem. Zato, kada se poveća dimenzija problema vidimo da ubrzanje dostiže 3,37x za Prewitt i čak 3,84x za 2. algoritam što je već mnogo bolje. Naravno, da je kompjuter na kome je rađeno testiranje imao više jezgara, videli bi još veće ubrzanje. Situacija se malo komplikuje ako uzmemo veći 5x5 filter za prewitt ili udaljenost=2 za 2. algoritam. Kod prewitt-a vidimo ubrzanje u odnosu na 3x3 od početka, gde i za najmanju sliku ubrzanje skače sa 2 na 2.5, a kod sledeće slike sa 3 na 3,92x što je izvanredno ubrzanje. To ubrzanje je, verovatno zbog slučajnosti okolnosti tokom testiranja, veće čak i od ubrzanja kod najveće slike. Isto tako, kod udaljenost=2 za 2. algoritam vidimo da iako takođe postoji početno ubrzanje u odnosu na udaljenost=1 kod manje slike, kod većih slika više nema nekih velikih razlika u ubrzanju. To se dešava jer je problem već dovoljno kompleksan da bi se postiglo neko „optimalno“ ubrzanje.

**Autor: Vuk Radmilović SV73-2020**