|  |
| --- |
| Paralelno programiranje |
| Detekcija ivica unutar slike |

|  |
| --- |
| Jelena Mikovic SV61/2020  6-13-2022 |



Sadržaj

[Analiza problema 2](#_Toc106165795)

[Koncept rešenja 2](#_Toc106165796)

[*Prewitt operator* 3](#_Toc106165797)

[*OneTBB zadaci* 4](#_Toc106165798)

[Programsko rešenje 4](#_Toc106165799)

[*Serijski program za detekciju ivica pomoću Prewitt operatora* 4](#_Toc106165800)

[*Paralelni program za detekciju ivica pomoću Prewitt operatora* 5](#_Toc106165801)

[*Crno-beli filter* 6](#_Toc106165802)

[*Serijski program za detekciju ivica metodom sa najužom okolinom* 6](#_Toc106165803)

[*Paralelni program za detekciju ivica metodom sa najužom okolinom* 7](#_Toc106165804)

[Analiza rezultata 8](#_Toc106165805)

[Testni slučajevi 8](#_Toc106165806)

[Tabelaran prikaz vremena u odnosu na veličinu filtera 10](#_Toc106165807)

[Grafički prikaz vremena u odnosu na cutoff 10](#_Toc106165808)

Analiza problema

Problematika kojom se ovaj rad bavi je detekcija ivica unutar slike. Detekcija ivica je tehnika obrade slike za pronalaženje granica objekata unutar slika. Radi tako što otkriva diskontinuitet u osvetljenosti. Detekcija ivica se koristi za segmentaciju slike i ekstrakciju podataka u oblastima kao što su obrada slike, kompjuterski vid i mašinski vid.

U osnovi digitalne obrade slike nalazi se operacija dvodimenzionalne diskretne konvolucije koja se svodi na množenje odgovarajućih elemenata dveju matrica i sumiranje rezultata. Očekivano rešenje se nalazi na slikama ispod (slika 1, slika 2).

U daljem tekstu se nalazi analiza četri načina obrade detektovanja ivica, zajedno sa test primerima i međusobnim poređenjem.

|  |  |
| --- | --- |
| C:\Users\Admin\AppData\Local\Microsoft\Windows\INetCache\Content.Word\450px-Bikesgray.jpg  *Slika 1* | C:\Users\Admin\AppData\Local\Microsoft\Windows\INetCache\Content.Word\450px-Bikesgray_prewitt.jpg  *Slika 2* |

Koncept rešenja

Implementirana su dva serijska načina obrade, prvenstveno pomoću Prewitt operatora, a zatim metodom sa najužom okolinom. Obe obrade su kasnije paralelizovane. Kod je kucan u programskom jeziku C++. Za paralelizovani deo je korišćena OneApi biblioteka, odnosno TBB zadaci, po uzoru na serijsko rešenje.

Obe obrade rade po principu klizanja matrice filtera preko svih podmatrica unete slike i obavljanje aritmetičkih operacija nad svakim ovako stvorenim parom.

Ulazna slika je .bmp formata sa RGB prostorom boja. Rezultat koda je crno bela slika, predstavljena kao matrica, gde beli pikseli predstavljaju ivice, a crni prostor između njih.

Prewitt operator

Prewitt operator izračunava gradijent intenziteta slike u svakoj tački, dajući pravac najvećeg mogućeg povećanja od svetlog do tamnog i brzinu promene u tom pravcu. Rezultat pokazuje koliko se "naglo" ili "glatko" slika menja u tom trenutku i koliko je verovatno da deo slike predstavlja ivicu.

Matematički, operator koristi dva 3×3 jezgra koja su konvolvirana sa originalnom slikom za izračunavanje aproksimacija derivata – jedno za horizontalne promene, a jedno za vertikalne.

|  |  |
| --- | --- |
|  | *Slika 3*.  Originalna slika |
| C:\Users\Admin\AppData\Local\Microsoft\Windows\INetCache\Content.Word\x.png | *Slika 4*.  Slika posle horizontalne konvulacije |
| C:\Users\Admin\AppData\Local\Microsoft\Windows\INetCache\Content.Word\y.png | *Slika 5*.  Slika posle vertikalne konvulacije |
|  | *Slika 6.*  Slika posle obe konvulacije  G = |Gx| + |Gy| |

OneTBB zadaci

Intel oneAPI Threading Building Blocks (oneTBB) je široko korišćena C++ biblioteka za paralelno programiranje sa deljenom memorijom zasnovano na zadacima na domaćinu. Paralelizacija u ovom projektu se vrši pomoću TBB task group-a, odnosno TBB zadataka.

## Programsko rešenje

Program je rađen u Visual Studio-u koristeći programski jezim C++ i biblioteku oneAPI (oneTBB).

Serijski program za detekciju ivica pomoću Prewitt operatora

Serijski program radi tako što prolazi redom kroz piksele slike, predstavljene kao matrica, i na njih primenjuje Prewitt filter. Kasnije se za svaki piksel određuje da li se boji u crno ili belo, odnosno da li je ivica ili ne.

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

{

for (int i = 1; i < width - 1; i++) {

for (int j = 1; j < height - 1; j++) {

int Gx = 0;

int Gy = 0;

int G = 0;

for (int m = 0; m < FILTER\_SIZE \* FILTER\_SIZE; m++) {

Gx += inBuffer[(j + filterHor[m]) \* width + (i + filterVer[m])] \* filterHor[m];

Gy += inBuffer[(j + filterHor[m]) \* width + (i + filterVer[m])] \* filterVer[m];

}

G = abs(Gx) + abs(Gy);

if (G > THRESHOLD) {

outBuffer[j \* width + i] = 255;

}

else {

outBuffer[j \* width + i] = 0;

}

}

}

}

Funkcija uzima parametre:

* inBuffer – bafer ulazne slike
* outBuffer – bafer izlazne slike
* width – širina slike
* height – visina slike

U potamljenom delu vidimo kako se računaju Gx i Gy po zadatoj formuli, kao i G kasnije.

Paralelni program za detekciju ivica pomoću Prewitt operatora

Paralelizovana implementacija Prewitt operatora se zasniva na group taskovima, koristeći oneTBB. Paralelizacija se izvraša do određenog odseka (cutoff), koji je u ovom slučaju 500, tako što se deli na 4 zadatka kojima se prosleđuju različiti parametri.

void filter\_parallel\_prewitt(int\* inBuffer, int\* outBuffer, int WIDTH, int HEIGHT, int width, int height, int startWidth, int startHeight)

{

task\_group g;

if (width < CUTOFF) {

parallel\_prewitt\_cufoff(inBuffer, outBuffer, WIDTH, HEIGHT, width, height, startWidth, startHeight);

}

else {

g.run([&] {filter\_parallel\_prewitt(inBuffer, outBuffer, WIDTH, HEIGHT, width / 2, height / 2, startWidth, startHeight); });

g.run([&] {filter\_parallel\_prewitt(inBuffer, outBuffer, WIDTH, HEIGHT, width / 2, height / 2, startWidth + width / 2, startHeight); });

g.run([&] {filter\_parallel\_prewitt(inBuffer, outBuffer, WIDTH, HEIGHT, width / 2, height / 2, startWidth, startHeight + height / 2); });

g.run([&] {filter\_parallel\_prewitt(inBuffer, outBuffer, WIDTH, HEIGHT, width / 2, height / 2, startWidth + width / 2, startHeight + height / 2); });

g.wait();

}

}

Funkcija uzima parametre:

* inBuffer – bafer ulazne slike
* outBuffer – bafer izlazne slike
* WIDTH – širina slike
* HEIGHT – visina slike
* width – širina podmatrice slike
* height – visina podmatrice slike
* startWidth – širina početka podmatrice slike
* startHeight – visina početka podmatrice slike

void parallel\_prewitt\_cufoff(int\* inBuffer, int\* outBuffer, int WIDTH, int HEIGHT, int width, int height, int startWidth, int startHeight)

{

if (startHeight == 0) startHeight += 1;

if (startWidth == 0) startWidth += 1;

for (int i = startWidth; i < width + startWidth; i++) {

if (i >= WIDTH - 1) break;

for (int j = startHeight; j < height + startHeight; j++) {

if (j >= HEIGHT - 1) break; … }

Funkcija ima identične parametre kao i sam paralelizovan deo koda. Prikazan je osenčen deo koji predstavlja kod koji se brine o tome da ivične podmatrice nikad ne pređu veličinu, odnosno visinu, originalne slike. Ostatak funkcije je identičan kao i sam serijski kod za Prewitt operator.

### Crno-beli filter

Da bi serijski program metodom sa najužom okolinom mogao da se odradi, prvo je potrebno da sliku prevedemo u matricu crnih i belih piksela predstavljene kao jedinice i nule.

void black\_and\_white(int\* inBuffer, int width, int height)

{

for (int i = 0; i < width; i++) {

for (int j = 0; j < height; j++) {

int value = inBuffer[j \* width + i];

if (value < THRESHOLD) {

inBuffer[j \* width + i] = 0;

}

else {

inBuffer[j \* width + i] = 1;

}

}

}

}

Funkcija uzima parametre:

* inBuffer – bafer ulazne slike
* outBuffer – bafer izlazne slike
* width – širina slike
* height – visina slike

Funkcija je krajnje jednostavna, za svaki piksel proverava da li je njegova vrednost manja od granice (threshold), koja je u ovom slučaju 128. Ako jeste njegova vrednost se postavlja na 0, ako nije na 1.

Serijski program za detekciju ivica metodom sa najužom okolinom

Nakon što je slika prevedena u matricu jedinica i nula, primenjuje se program za detekciju ivica metodom sa najužom okolinom. Započinje se pretraga okoline svake tačke, tako što se poredi sa susednim tačkama (tačke: levo, desno, iznad, ispod i dijagonalno na sve strane).

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

{

for (int i = 1; i < width - 1; i++) {

for (int j = 1; j < height - 1; j++) {

int P = 0;

int O = 1;

int pixel = 0;

for (int m = 0; m < FILTER\_SIZE \* FILTER\_SIZE; m++) {

if (m == FILTER\_SIZE \* FILTER\_SIZE/2)

continue;

if (inBuffer[(j + filterHor[m]) \* width + (i + filterVer[m])] == 1) P = 1;

if (inBuffer[(j + filterHor[m]) \* width + (i + filterVer[m])] == 0) O = 0;

}

pixel = abs(P - O);

if (pixel == 1) {

outBuffer[j \* width + i] = 255;

}

else {

outBuffer[j \* width + i] = 0;

}

}

}

}

Funkcija uzima parametre:

* inBuffer – bafer ulazne slike
* outBuffer – bafer izlazne slike
* width – širina slike
* height – visina slike

Potamljen deo proverava da li je reč o tački, odnosno pikselu, na kom smo trenutno i u tom slučaju ga preskače.

Paralelni program za detekciju ivica metodom sa najužom okolinom

Paralelini program za ovu metodu je dosta sličan programu za paralelnu obradu sa Prewitt filterom, a i deo nakon odseka je rađen po istom principu sa istom brigom o tome da ivične podmatrice nikad ne pređu veličinu, odnosno visinu, originalne slike.

Analiza rezultata

Rezultati variraju u odnosu na brojne faktore, kao što je veličina filtera, veličina slike, karakteristike računara, veličina cutoff-a. U toku testiranja je vršen veliki broj varijacija i u daljem tekstu se nalaze zapažanja i zaključci.

Specifikacija računara na kom je testirano:

* Processor: AMD Ryzen 3 3200U with Radeon Vega Mobile Gfx 2.60 GHz
* RAM: 4,00 GB (3,21 GB usable)
* System type: 64-bit operating system, x64-based processor

Testni slučajevi

Testiran je određen broj slika, gde je svaka karakteristična po nečemu. Neke su crno-bele, neke dosta tamne, nekima nisu lako uočljive ivice, a negde su krajnje očigledne. Rezultati se nalaze na tabeli ispod. Bitno zapažanje je da su vremena znatno brža na ovim slikama, jer su one podosta manje od prvobitno bila zadate u ovom projektu. Čak i ako su slike premale, obrada bez paralelizacije postaje brža.

|  |  |
| --- | --- |
| C:\Users\Admin\Desktop\color.bmp  *Slika 10* | C:\Users\Admin\Desktop\Windows\Windows\outputParallelEdge.bmp  *Metodom sa najužom okolinom, koristeći 3x3 filter, cutoff 500, vreme 37ms, a za paralelnu verziju 20ms* |
| C:\Users\Admin\AppData\Local\Microsoft\Windows\INetCache\Content.Word\color.bmp  *Slika 7* | C:\Users\Admin\AppData\Local\Microsoft\Windows\INetCache\Content.Word\outputSerialPrewitt.bmp  *Prewitt metod, koristeći 7x7 filter, cutoff 300, vreme 709ms, a za paralelnu verziju 525ms* |
| *Slika 8* | *Metodom sa najužom okolinom, koristeći 3x3 filter, cutoff 300, vreme 75ms, a za paralelnu verziju 47ms* |
| C:\Users\Admin\AppData\Local\Microsoft\Windows\INetCache\Content.Word\color.bmp  *Slika 9* | *Prewitt metod, koristeći 3x3 filter, cutoff 300, vreme 92ms, a za paralelnu verziju 178ms* |

Tabelaran prikaz vremena u odnosu na veličinu filtera

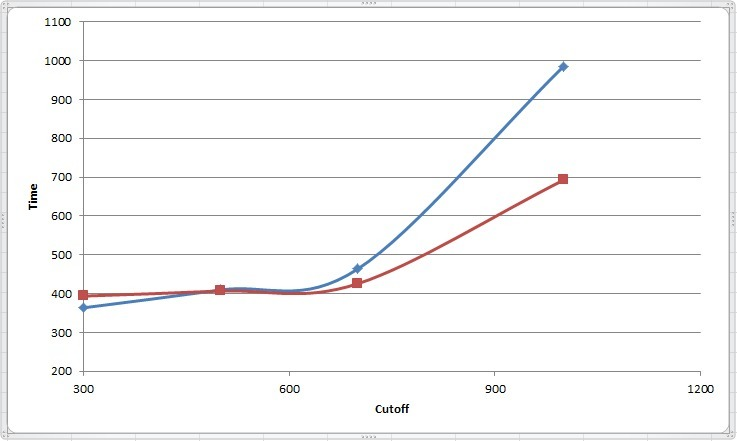
Na tebeli ispod je prikazana zavisnost vremena od veličine filtera. Bitno je napomenuti da su filteri prošireni sa vrednostima -1, 0 i 1, kao i da je odsek, cutoff, bio 300.

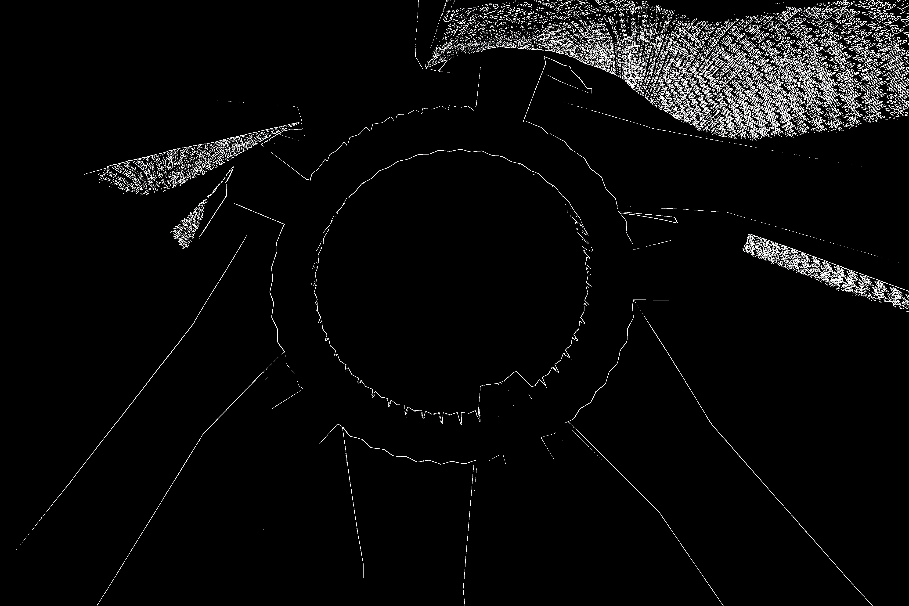
|  |  |  |  |
| --- | --- | --- | --- |
| Filter: | 3x3 | 5x5 | 7x7 |
| Prewitt filter | 1344,48ms | 2192,07ms | 3801,57ms |
| Paralelni Prewitt filter | 379,983ms | 845,062ms | 1634,88ms |
| Metodom sa najužom okolinom | 939,23ms | 2019,48ms | 3286,84ms |
| Paralelni metodom sa najužom okolinom | 366,824ms | 751,849ms | 1336,64ms |

Na slikama 3, 4 i 5 su redom prikazani rezultati korišćenjem Prewitt filtera za obradu slike kada je jezgro veličine 3, 5 i 7. Kao što i sa tabele se može zaključiti, što je veći filter to duže traje sam program, međutim dobijamo detaljniju sliku.

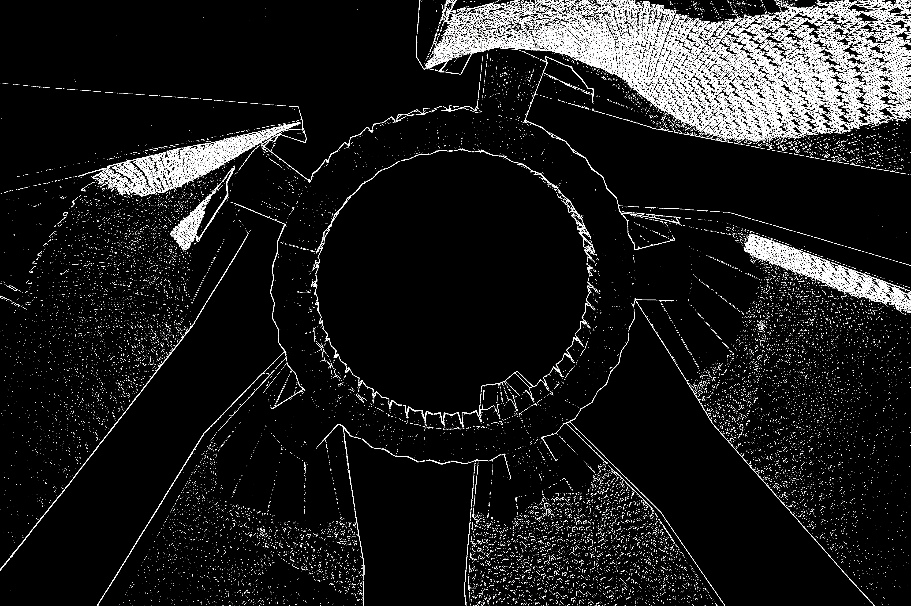
Grafički prikaz vremena u odnosu na cutoff

Na grafiku je prikazana zavisnost vremena izvršavanja obrade od cutoff-a. Što je veći cutoff, program se sporije izvršava, međutim sve ispod 300 nije prikazivalo značajne promene. Linija sa manjim kvadratićima predstavlja metod obrade sa Perwitt operatorom, a sa većim metod sa najužom okolinom.

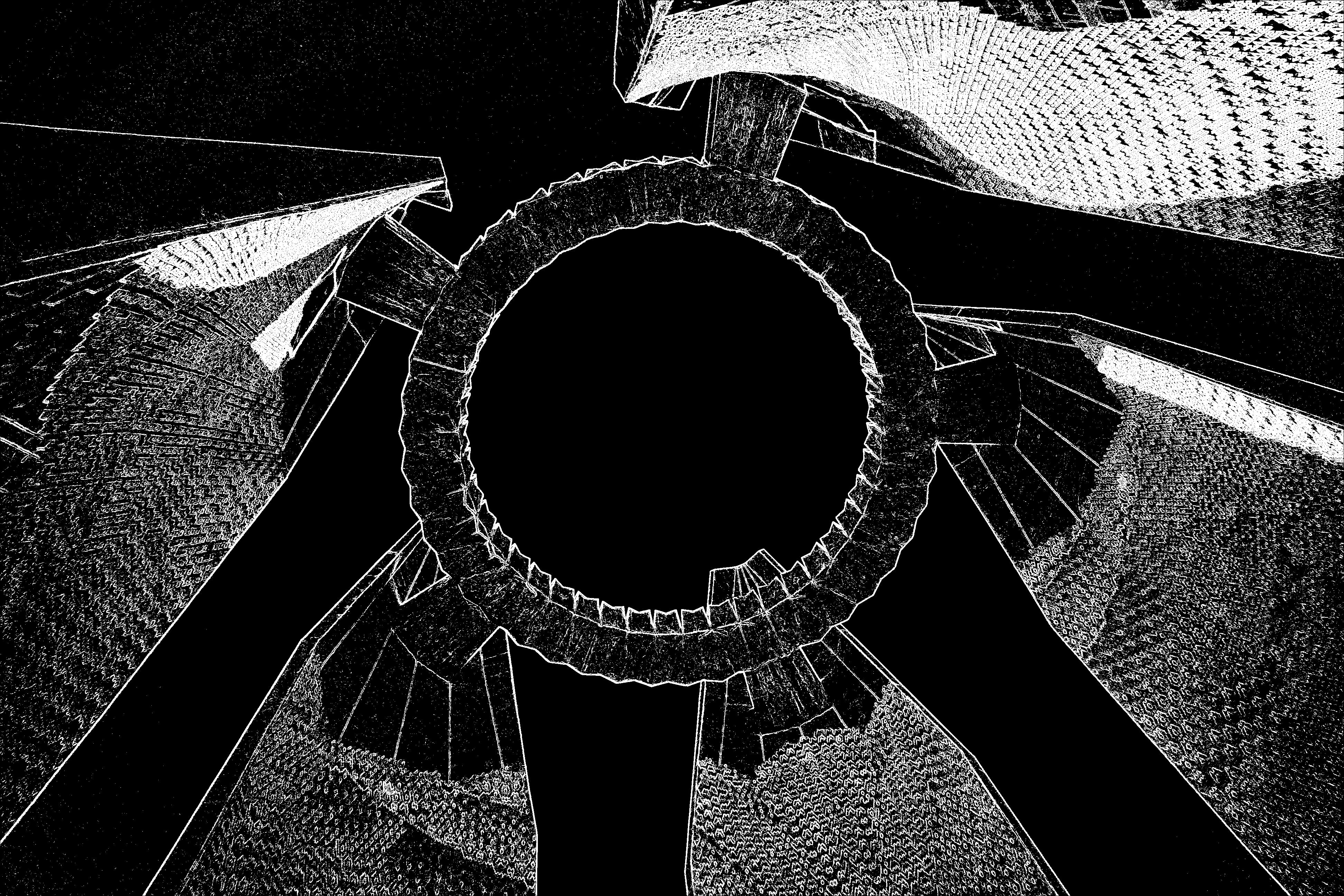




*Slika 11 – Prewitt 3x3*



*Slika 12 – Prewitt 5x5*



*Slika 13 – Prewitt 7x7*