



Universitatea Politehnica București  
Facultatea de Automatică și Calculatoare  
Departamentul de Automatică și Ingineria Sistemelor

# Image Smoothing - Convolution Mask

Student  
Cruțeru Raluca-Elena, 331AA

Prof. Hossu Andrei

București, 2024

# Cuprins

<b>1</b>	<b>Descriere Algoritm Gaussian Smoothing</b>	<b>1</b>
<b>2</b>	<b>Principii de Bază</b>	<b>2</b>
<b>3</b>	<b>Descriere structurala</b>	<b>3</b>
<b>4</b>	<b>Descriere arhitecturala</b>	<b>4</b>
<b>5</b>	<b>Utilizare</b>	<b>5</b>
<b>6</b>	<b>Rezultate</b>	<b>6</b>

# 1 Descriere Algoritm Gaussian Smoothing

Algoritmul Gaussian Smoothing a fost implementat în `jarsmoothing.jar` pentru a realiza procesul de "netezire".

## Procesul de Convoluție

În etapa de convoluție, imaginea finală se obține aplicând un filtru, cunoscut sub denumirea de "kernel," peste matricea originală a imaginii. Kernelul este plasat succesiv pe fiecare pixel al imaginii, iar valorile corespondente sunt înmulțite cu cele din matricea originală. Produsele obținute sunt apoi sumate, furnizând valoarea pixelului corespunzător în imaginea finală.

## Kernel Gaussian

Kernelul Gaussian, folosit în acest context, este o matrice de pixeli calculată cu ajutorul formulei:

$$G(x, y) = \frac{1}{\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$$

unde  $x$  reprezintă distanța de la origine în axa orizontală,  $y$  este distanța de la origine pe axa verticală, iar  $\sigma$  reprezintă deviația standard a distribuției Gauss.

În procesul de convoluție, se aplică kernelul Gaussian asupra imaginii originale, iar rezultatul influențează intensitatea pixelilor din imaginea finală. Această etapă contribuie la crearea unei reprezentări vizuale "netezite," eliminând zgomotul și evidențiind detaliile semnificative ale imaginii.

Întrucât proiectul este bazat pe operații de manipulare a imaginilor, timpul de procesare este monitorizat cu ajutorul clasei `Tmr`, care oferă funcționalități de cronometrare.

De asemenea, pentru interacțiunea cu utilizatorul, proiectul include clasa `Handler`, care se ocupă de gestionarea parametrilor din consola.

Această abordare permite utilizatorilor să aplice netezirea Gaussiană asupra imaginilor BMP și să obțină rezultate vizuale îmbunătățite prin reducerea zgomotului și accentuarea detaliilor semnificative.

## 2 Principii de Bază

**Modularitate:** Programul este structurat în module sau clase distincte, fiecare având o funcționalitate specifică, cum ar fi citirea imaginilor, aplicarea convoluției sau scrierea imaginilor. Această abordare modulară facilitează dezvoltarea, testarea și întreținerea.

**Orientat pe Obiecte:** Programarea orientată pe obiecte (OOP) a fost adoptată pentru organizarea codului în clase și obiecte. Acest lucru permite abstractizarea funcționalităților și îmbunătățește reutilizarea codului.

**Extensibilitate:** Prin intermediul claselor abstracte și interfețelor, programul permite extinderea funcționalității prin adăugarea de noi implementări pentru citire, scriere sau operații de convoluție. Astfel, utilizatorii pot adăuga sau modifica ușor funcționalități în viitor.

## 3 Descriere structurala

**Handler - Gestionarea Liniei de Comandă** Clasa Handler este responsabilă de preluarea și validarea parametrilor introdusi de utilizator prin intermediul liniei de comandă. Acesta utilizează Scanner pentru a colecta informațiile și lansează o excepție (ExceptionLine) în caz de erori.

**ImageGet și ImageSet - Citirea și Scrierea Imaginilor** Interfețele ImageGet și ImageSet definesc metodele pentru citirea și scrierea imaginilor. Clasele BmpGet și BmpSet implementează aceste interfețe, utilizând ImageMatrix pentru reprezentarea datelor imaginilor.

**AbstractConvolution și Convolution - Convoluție Imagini** Clasa abstractă AbstractConvolution oferă structura de bază pentru operații de convoluție, în timp ce Convolution implementează efectiv operația de convoluție. Acesta utilizează o mască de convoluție și un divizor pentru a aplica efectul dorit.

Convoluția în prelucrarea imaginilor este o operație matematică utilizată pentru a aplica filtre sau efecte asupra imaginilor. Aceasta implică suprapunerea unei măști (sau kernel) peste imaginea originală și calcularea sumei produselor elementelor măștii și intensității pixelilor corespondenți din imaginea originală.

**Smoothing - Convoluție cu Efect de Netezire Gaussiană** Clasa Smoothing extinde Convolution pentru a atinge un nivel mai înalt de moștenire și particularizează valorile kernelului și ale divizorului, aplicând o mască de convoluție specifică pentru a realiza un efect de netezire gaussiană asupra imaginii.

Netezirea Gaussiană este o tehnică de prelucrare a imaginilor utilizată pentru a reduce zgomotul și a netezi detaliile. Se bazează pe aplicarea unui filtru Gaussian, o funcție matematică care favorizează valorile din centrul măștii și reduce ponderea valorilor la distanță.

**Tmr - Măsurarea Timpului** Clasa Tmr oferă funcționalități de măsurare a timpului, utilizată pentru a evalua performanța diferitelor operații ale programului - va calcula timpul de procesare al operațiilor : citire, scriere, convolutia propriu-zisă a imaginii.

**ImageProcessingMain - Clasa Principală** Clasa principală ImageProcessingMain orcheștrează funcționalitățile întregului program. Acesta primește parametri de la Handler, apoi utilizează BmpGet pentru a citi imaginea,

## 4 Descriere arhitecturala

Descrierea arhitecturală a codului se bazează pe principii de programare orientată pe obiect, inclusiv pe conceptele de polimorfism, moștenire și abstractizare.

**Polimorfism:** În proiect, polimorfismul este evident în utilizarea interfețelor și a claselor abstracte. De exemplu, `ImageGet`, `ImageSet`, și `AbstractConvolution` definesc funcționalități generice care sunt implementate specific în clasele concrete. Metoda `convolute` din clasa `Convolution` este un exemplu de polimorfism, deoarece este suprascrisă în clasa derivată `Smoothing`.

**Moștenire:** Moștenirea este folosită pentru a specializa funcționalitatea în clasa `Smoothing`, care extinde clasa `Convolution`. Astfel, funcționalitatea generică a convoluției este îmbunătățită și specializată pentru netezire.

**Abstractizare:** Utilizarea claselor și interfețelor abstracte precum `AbstractConvolution` și `ImageSet` demonstrează conceptul de abstractizare, unde detalii specifice sunt ascunse pentru a oferi o interfață simplificată și generică.

**Encapsulare:** Clasele sunt concepute cu encapsulare, unde atributele sunt private și se accesează prin metode publice. De exemplu, metodele `getKernel` și `setKernel` din clasa `Smoothing` demonstrează encapsularea atributelor kernel-ului.

**Interfețe:** Utilizarea interfețelor, cum ar fi `ImageGet` și `ImageSet`, ajută la definirea unui contract comun pentru toate clasele care le implementează, facilitând astfel extensibilitatea și schimbul de componente.

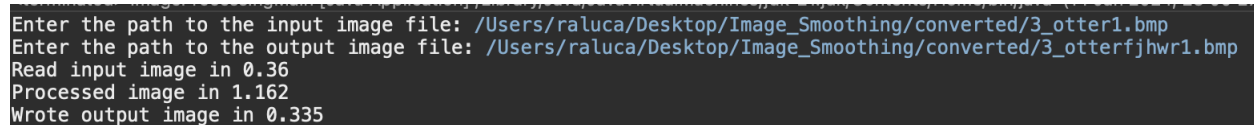
**Gestionarea Excepțiilor:** Prin crearea clasei `BmpException` și extinderea clasei `ExceptionLine`, gestionarea excepțiilor este realizată în mod corespunzător, oferind o modalitate de tratare a erorilor în cadrul aplicației.

## 5 Utilizare

Pentru a utiliza acest cod, este necesară adăugarea bibliotecii commons-cli-1.4 în cadrul build-ului, pentru a asigura funcționarea corectă a programului.

Interacțiunea cu programul se realizează prin intermediul consolei, unde utilizatorul este ghidat să introducă calea către fișierul BMP pe care dorește să îl modifice și calea către locația de salvare a rezultatului.

Ca rezultat, vor fi afișați timpii de execuție pentru fiecare dintre operații, oferind utilizatorului informații despre durata fiecărei etape a procesului.



```
Enter the path to the input image file: /Users/raluca/Desktop/Image_Smoothing/converted/3_otter1.bmp
Enter the path to the output image file: /Users/raluca/Desktop/Image_Smoothing/converted/3_otterfjhw1.bmp
Read input image in 0.36
Processed image in 1.162
Wrote output image in 0.335
```

Figura 5.1: Rezultate rulare program

## 6 Rezultate

Ruland programul pentru imaginea flower.bmp se obtine:



Timpul de procesare în acest proiect poate varia în funcție de câteva aspecte, inclusiv dimensiunea imaginii, complexitatea operațiilor aplicate, resursele hardware ale sistemului și eficiența algoritmilor utilizați. Iată câteva factori care pot influența timpul de procesare:

**Dimensiunea Imaginii:** Imaginile mai mari vor necesita mai mult timp pentru a fi procesate, deoarece implicațiile algoritmilor de manipulare a imaginilor sunt direct proporționale cu dimensiunea imaginii.

**Complexitatea Operațiilor:** Algoritmii precum Gaussian Smoothing sunt destul de eficienți, dar complexitatea lor poate influența timpul de execuție. Cu cât operațiile de prelucrare sunt mai complexe, cu atât timpul de procesare poate crește.

**Eficiența Implementării:** Implementarea algoritmilor poate varia în eficiență. Un cod optimizat și bine scris poate duce la timpi de execuție mai rapizi, în timp ce o implementare mai ineficientă poate încetini procesul.

**Resurse Hardware:** Performanța sistemului de calcul pe care rulezi programul este un factor important. Sistemele mai puternice, cu procesoare mai rapide și cu mai multă memorie, pot prelucra imaginile mai repede.

**Numărul de Operații Aplicate:** Dacă aplici mai multe operații sau filtre pe imagine, timpul de procesare va crește în mod corespunzător.