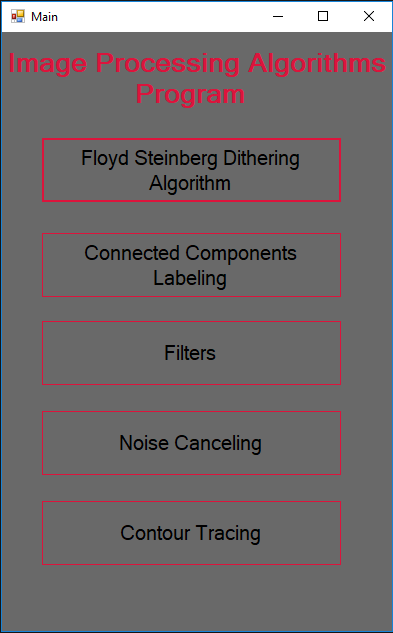
Algoritmi pentru Prelucrarea

Imaginilor

Student:

Miucin Sebastian

Aplicatia consta in 5 algoritmi pentru prelucrarea imaginilor:



1. Floyd Steinberg Dithering Algorithm

Floyd-Steinberg dithering este un algoritm de dithering de imagine publicat pentru prima data in 1976 de Robert W. Floyd si Louis Steinberg. Acesta este utilizat în mod obișnuit de software-ul de manipulare a imaginii, de exemplu atunci când o imagine este convertită în format GIF care este limitată la maximum 256 de culori.

Algoritmul realizează dithering folosind difuzia de erori, adică împinge (adaugă) eroarea de cuantizare reziduală a unui pixel pe pixelii săi vecini, care urmează să fie tratate mai târziu. Se extinde datoria în funcție de distribuție (prezentată ca o hartă a pixelilor vecini): Floyd-Steinberg dithering este un algoritm de dithering de imagine publicat pentru prima data in 1976 de Robert W. Floyd si Louis Steinberg. Acesta este utilizat în mod obișnuit de software-ul de manipulare a imaginii, de exemplu atunci când o imagine este convertită în format GIF care este limitată la maximum 256 de culori.

Algoritmul realizează dithering folosind difuzia de erori, adică împinge (adaugă) eroarea de cuantizare reziduală a unui pixel pe pixelii săi vecini, care urmează să fie tratate mai târziu. Se extinde datoria în funcție de distribuție (prezentată ca o hartă a pixelilor vecini).

Algoritm:

**for each** y from top to bottom

**for each** x from left to right

oldpixel := pixel[x][y]

newpixel := find\_closest\_palette\_color(oldpixel)

pixel[x][y] := newpixel

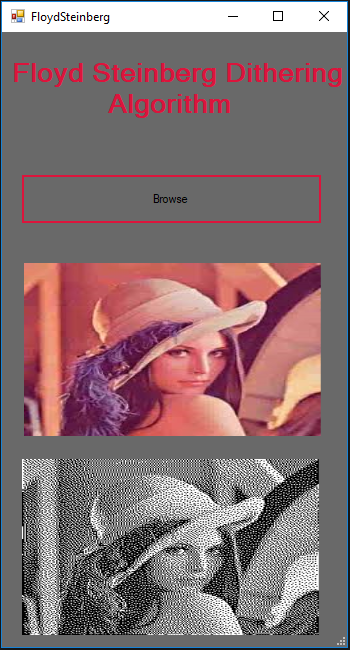
quant\_error := oldpixel - newpixel

pixel[x + 1][y ] := pixel[x + 1][y ] + quant\_error \* 7 / 16

pixel[x - 1][y + 1] := pixel[x - 1][y + 1] + quant\_error \* 3 / 16

pixel[x ][y + 1] := pixel[x ][y + 1] + quant\_error \* 5 / 16

pixel[x + 1][y + 1] := pixel[x + 1][y + 1] + quant\_error \* 1 / 16



1. Connected Components Labeling

Etichetarea componentelor conectate (analiza componentelor conectate, extragerea blob-urilor, etichetarea regiunilor, descoperirea blobului sau extragerea regiunilor) este o aplicație algoritmică a teoriei grafurilor, unde subseturile de componente conectate sunt etichetate în mod unic pe baza unui euristic dat. Etichetarea componentelor conectate nu trebuie confundată cu segmentarea.

Etichetarea componentelor conectate este utilizată în viziunea computerului pentru a detecta regiunile conectate în imagini digitale binare, deși pot fi procesate și imagini color și date cu dimensiuni mai mari. Atunci când sunt integrate într-un sistem de recunoaștere a imaginilor sau interfață interacțiune om-calculator, etichetarea componentelor conectate poate opera pe o varietate de informații. Extracția de blob este efectuată, în general, pe imaginea binară rezultată dintr-o etapă de prag, dar poate fi aplicabilă și imaginilor de culoare gri și color. Bloburile pot fi numărate, filtrate și urmărite.

Extracția extracției este asociată, dar distinctă de detectarea blobului.

Algoritm:

**algorithm** ***TwoPass***(data)

linked = []

labels = structure with dimensions of data, initialized with the value of Background

*First pass*

**for** row **in** data:

**for** column **in** row:

**if** data[row][column] **is not** Background

neighbors = connected elements with the current element's value

**if** neighbors **is** empty

linked[NextLabel] = ***set*** containing NextLabel

labels[row][column] = NextLabel

NextLabel += 1

**else**

*Find the smallest label*

L = neighbors labels

labels[row][column] = ***min***(L)

**for** label **in** L

linked[label] = ***union***(linked[label], L)

*Second pass*

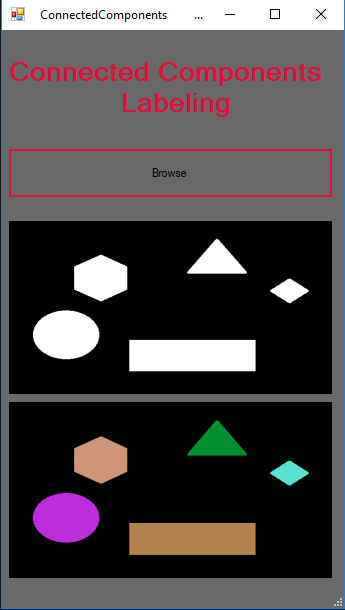
**for** row **in** data

**for** column **in** row

**if** data[row][column] **is not** Background

labels[row][column] = ***find***(labels[row][column])

**return** labels



1. Filtre (Low Pass , High Pass , Gaussian)

LPF:

Un filtru trece-jos (LPF) este un filtru care transmite semnale cu o frecvență mai mică decât o anumită frecvență de cutoff și atenuează semnalele cu frecvențe mai mari decât frecvența cutoff. Răspunsul exact al frecvenței filtrului depinde de designul filtrului. Filtrul este denumit uneori un filtru de înaltă tensiune sau un filtru triplu în aplicații audio. Un filtru trece-jos este complementul unui filtru de trecere.

HPF:

Un filtru de trecere superioară (HPF) este un filtru electronic care transmite semnale cu o frecvență mai mare decât o anumită frecvență de cutoff și atenuează semnalele cu frecvențe mai mici decât frecvența cutoff.

Gaussian:

În cazul filtrului de tip Gauss, coeficienţii de frecvenţă nu sunt tăiaţi brusc, ci este folosit un proces de suprimare mai netedă. Acest proces ţine cont şi de faptul că DFT a unei funcţii de tip Gauss este tot o funcţie de tip Gauss.

Algoritm:

  intialising standard deviation to 1.0

    double sigma = 1.0;

    double r, s = 2.0 \* sigma \* sigma;

    // sum is for normalization

    double sum = 0.0;

    // generating 5x5 kernel

    for (int x = -2; x <= 2; x++)

    {

        for(int y = -2; y <= 2; y++)

        {

            r = sqrt(x\*x + y\*y);

            GKernel[x + 2][y + 2] =

                     (exp(-(r\*r)/s))/(M\_PI \* s);

            sum += GKernel[x + 2][y + 2];

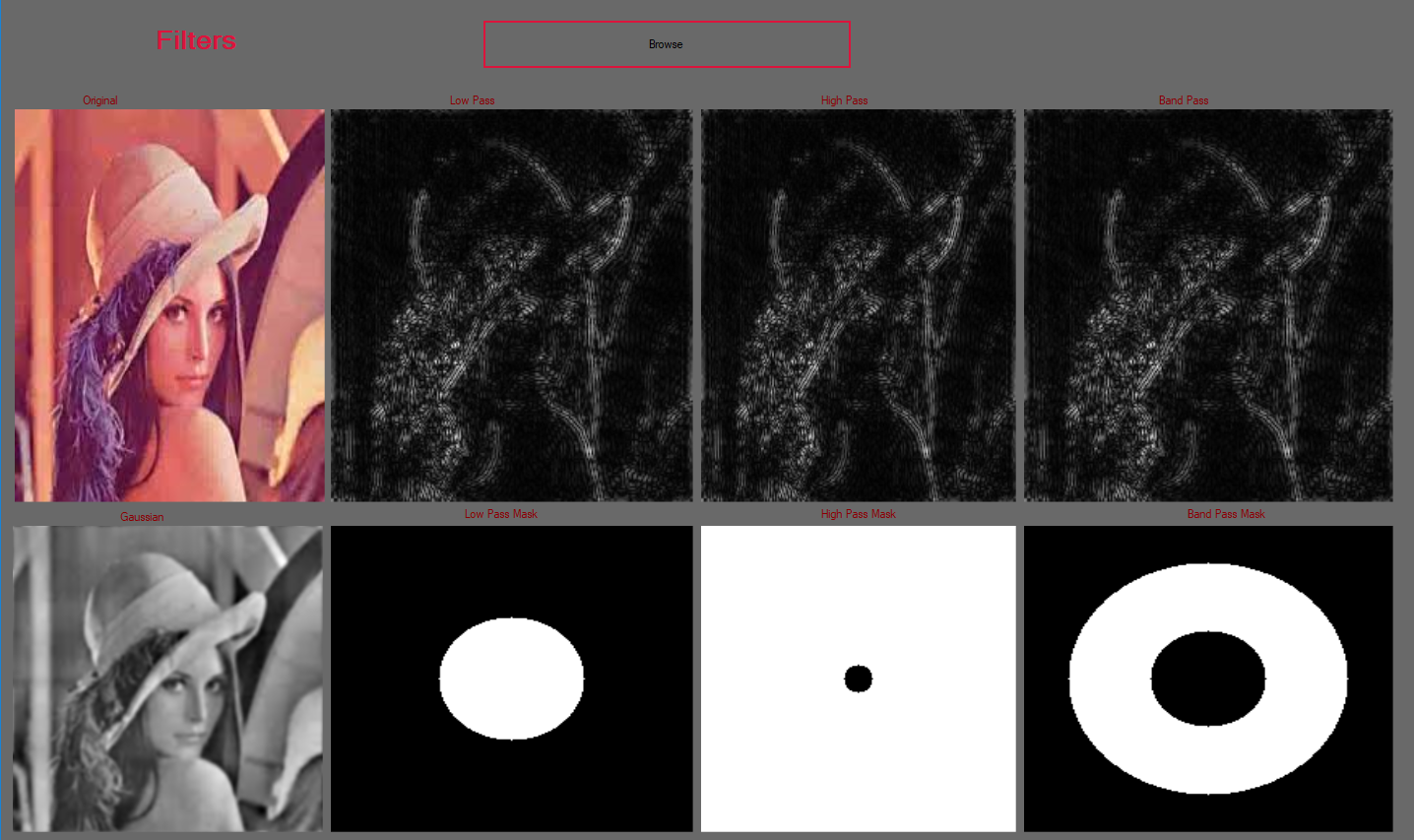
        }

    }

    for (int i = 0; i < 5; ++i)

        for (int j = 0; j < 5; ++j)

            gKernel[i][j] /= sum;



1. Noise reduction

În procesarea imaginilor, o estompare Gaussiană (cunoscută și sub denumirea de netezire Gaussiană) este rezultatul neclarității imaginii printr-o funcție Gaussiană. Este un efect utilizat pe scară largă în software-ul grafic, de obicei pentru a reduce zgomotul imaginii și pentru a reduce detaliile. Efectul vizual al acestei tehnici de estompare este o neclaritate netedă asemănătoare cu cea a vizualizării imaginii printr-un ecran translucid, diferit în mod distinct de efectul bokeh produs de o lentilă în afara focului sau de umbra unui obiect aflat sub iluminare obișnuită. Gălărirea Gaussiană este, de asemenea, utilizată ca o etapă de pre-procesare a algoritmilor de vizualizare a computerului pentru a îmbunătăți structurile de imagine la scări diferite.

Algoritm:

public static PGM ApplyMeanFilter(PGM inputImage, int iterationCount)

{

PGM outputImage = new PGM(inputImage.Size.Width, inputImage.Size.Height);

for (int iteration = 0; iteration < iterationCount; iteration++)

{

for (int x = 0; x < inputImage.Size.Width - 1; x++)

{

for (int y = 0; y < inputImage.Size.Height - 1; y++)

{

List<int> validNeighbors = inputImage.GetAllNeighbors(x, y);

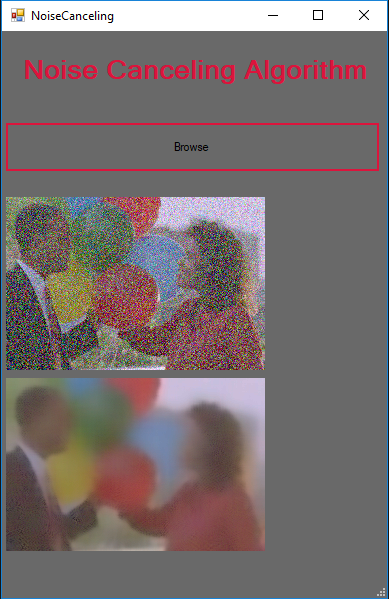
double averageIntensity = validNeighbors.Average()

outputImage.Pixels[x, y] = (short)averageIntensity;

}

}

inputImage = outputImage.Clone();} return (outputImage); }



1. Contour Tracing

Trasarea la frontieră (cunoscută și sub denumirea de urmărire a contururilor) a unei regiuni digitale binare poate fi considerată ca o tehnică de segmentare care identifică pixelii de graniță ai regiunii digitale. Trasarea la frontiere este un prim pas important în analiza acestei regiuni. Algoritmul algoritmului de urmărire a pătrunderii, algoritmul Moore-Neighbour Tracking și algoritmul lui Theo Pavlidis au fost folosite pentru urmărirea limitelor. Cu o reprezentare completă a unei celule digitale a unei imagini digitale, coordonatele punctului de frontieră pot fi extrase din acea imagine digitală urmând un algoritm creat de Dr. Kovalevsky folosind diviziunea modulo și căutări simple de pixeli.

Algoritm:

**struct** Point

{

int x; *// col*

int y; *// row*

P(int a , int b)

{

x = a;

y = b;

};

};

**const** int width = 6;

**const** int height = 10;

int[height][width] image = { { 0 , 0 , 0 , 0 , 0 , 0 } ,

{ 0 , 0 , 1 , 1 , 1 , 1 } ,

{ 0 , 0 , 1 , 1 , 1 , 1 } ,

{ 0 , 1 , 1 , 1 , 1 , 1 } ,

{ 1 , 1 , 1 , 1 , 1 , 0 } ,

{ 1 , 1 , 1 , 1 , 1 , 0 } ,

{ 0 , 0 , 1 , 1 , 1 , 0 } ,

{ 0 , 0 , 1 , 1 , 1 , 0 } ,

{ 0 , 0 , 1 , 1 , 1 , 0 } ,

{ 0 , 0 , 0 , 0 , 0 , 0 } };

Point startPoint;

**for** ( int row = 0 ; row < height ; row++ )

{

**for** ( int col = 0 ; col < width ; col++ )

{

**if** ( image[row][col] == 1 )

{

startPoint.x = col;

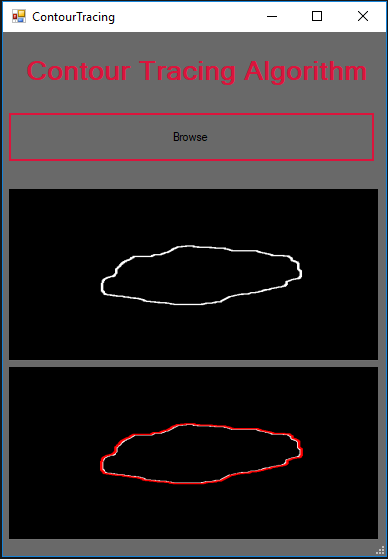
startPoint.y = row;

**return**;

}

}

}



Bibliogragie:

<https://en.wikipedia.org/wiki/Gaussian_blur>

<https://en.wikipedia.org/wiki/Gaussian_filter>

<https://en.wikipedia.org/wiki/High-pass_filter>

<https://en.wikipedia.org/wiki/Floyd%E2%80%93Steinberg_dithering>