**Nome: Lucas Miranda Mendonça Rezende**

N. USP: 12542838

**relatório.doc Continuando filtros digitais**

**Questão 1.1:**

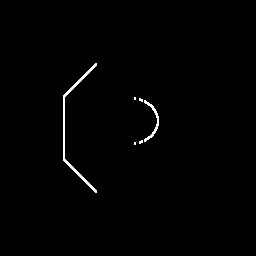
mit.tif (non-sep):



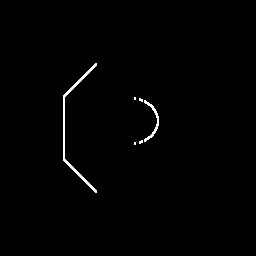
mit.tif (sep):



octagon.tif (non-sep):



octagon.tif (sep):



**Questão 1.2:**

**static public ImageAccess detectEdgeHorizontal\_Separable(ImageAccess input) {**

**int nx = input.getWidth();**

**int ny = input.getHeight();**

**ImageAccess out = new ImageAccess(nx, ny);**

**double rowin[] = new double[nx];**

**double rowout[] = new double[nx];**

**for (int y = 0; y < ny; y++) {**

**input.getRow(y, rowin);**

**doAverage3(rowin, rowout);**

**out.putRow(y, rowout);**

**}**

**double colin[] = new double[ny];**

**double colout[] = new double[ny];**

**for (int x = 0; x < nx; x++) {**

**out.getColumn(x, colin);**

**doDifference3(colin, colout);**

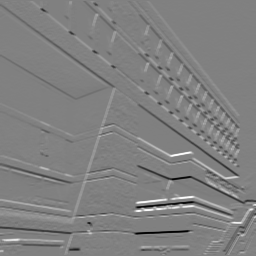
**out.putColumn(x, colout);**

**}**

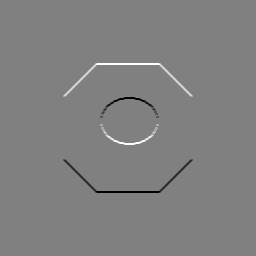
**return out;**

**}**

mit.tif (sep):



octagon.tif (sep):



**Questão 1.3:**

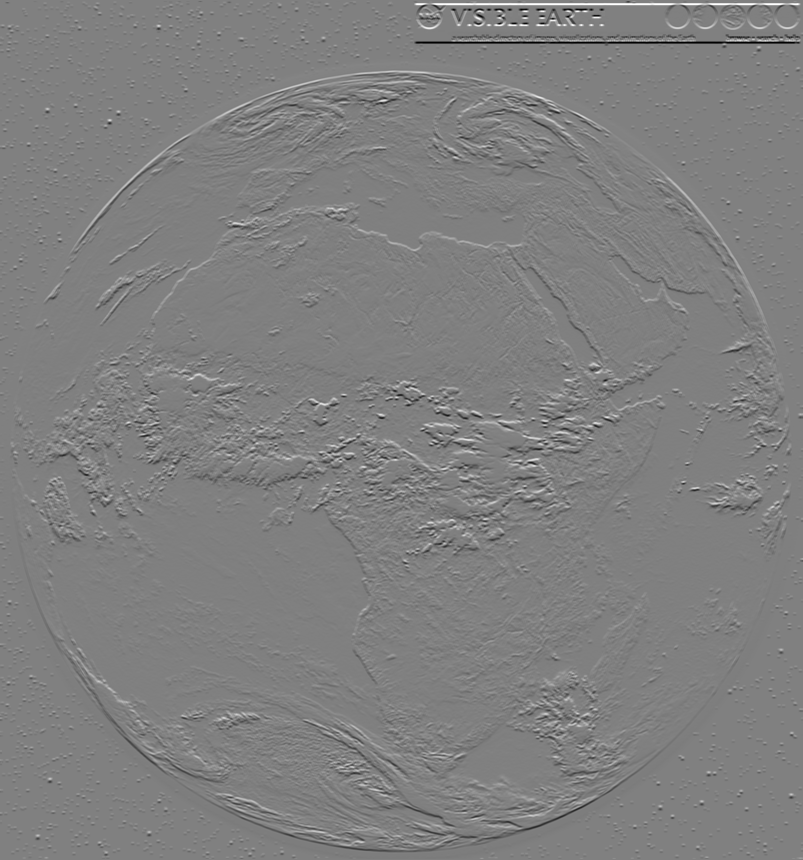
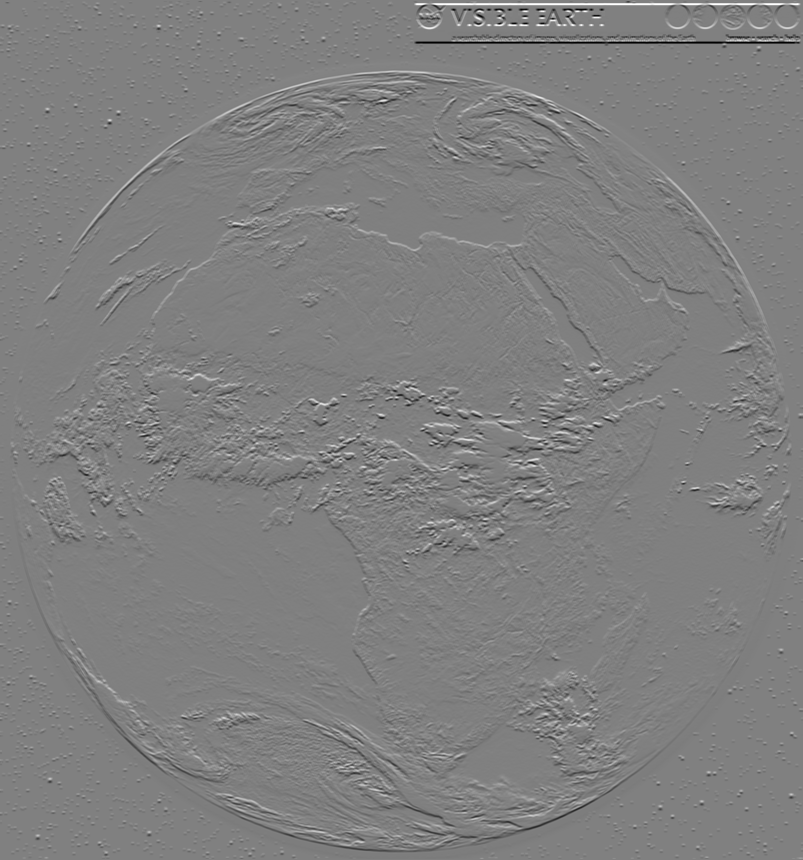
Detector de Bordas Horizontal em africa.tif

|  | Tempo | Média | Minimo | Maximo |
| --- | --- | --- | --- | --- |
| versão não-separável | 47ms | 0 | -87.06 | 85.23 |
| versão separável | 33ms | 0 | -107.19 | 119.11 |

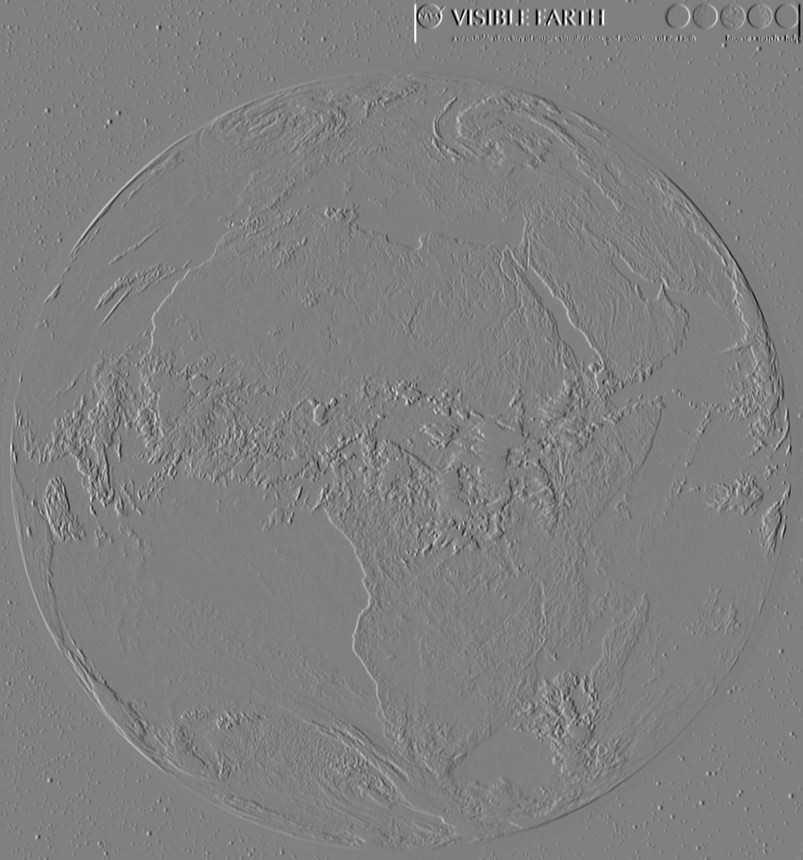
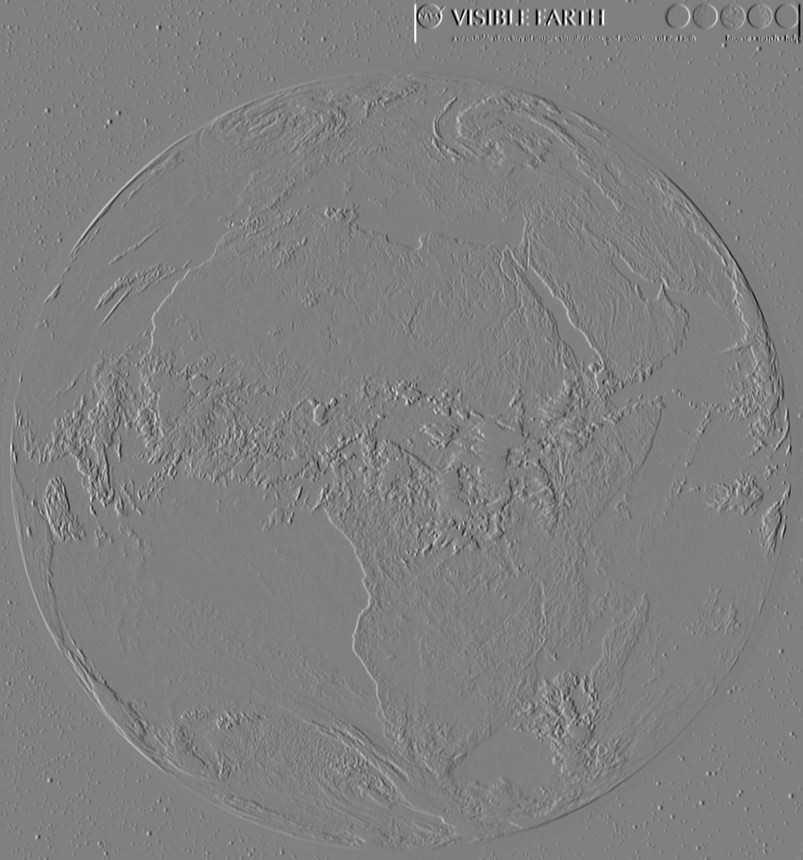
Comparação:

Embora semelhantes, a imagem da versão não separável parece que é mais clara, tem contraste maior. A versão não separável é a versão mais computacionalmente cara (+42%).

Horizontal (Sep/Non-Sep)



Vertical (Sep/Non-Sep)



Insira a parte do código escrito para vertical edge detector (mit.tif)

static public ImageAccess detectEdgeVertical\_NonSeparable(ImageAccess input) {

int nx = input.getWidth();

int ny = input.getHeight();

double arr[][] = new double[3][3];

double pixel;

ImageAccess out = new ImageAccess(nx, ny);

for (int x = 0; x < nx; x++) {

for (int y = 0; y < ny; y++) {

input.getNeighborhood(x, y, arr);

pixel = arr[2][0]+arr[2][1]+arr[2][2]-arr[0][0]-arr[0][1]-arr[0][2];

pixel = pixel / 6.0;

out.putPixel(x, y, pixel);

}

}

return out;

}

static public ImageAccess detectEdgeVertical\_Separable(ImageAccess input) {

int nx = input.getWidth();

int ny = input.getHeight();

ImageAccess out = new ImageAccess(nx, ny);

double rowin[] = new double[nx];

double rowout[] = new double[nx];

for (int y = 0; y < ny; y++) {

input.getRow(y, rowin);

doDifference3(rowin, rowout);

out.putRow(y, rowout);

}

double colin[] = new double[ny];

double colout[] = new double[ny];

for (int x = 0; x < nx; x++) {

out.getColumn(x, colin);

doAverage3(colin, colout);

out.putColumn(x, colout);

}

return out;

}

Insira a parte do código escrito para horizontal edge detector (mit.tif)

static public ImageAccess detectEdgeHorizontal\_NonSeparable(ImageAccess input) {

int nx = input.getWidth();

int ny = input.getHeight();

double arr[][] = new double[3][3];

double pixel;

ImageAccess out = new ImageAccess(nx, ny);

for (int x = 0; x < nx; x++) {

for (int y = 0; y < ny; y++) {

input.getNeighborhood(x, y, arr);

pixel= arr[0][2]+arr[1][2]+arr[2][2]-arr[0][0]-arr[1][0]-arr[2][0];

pixel = pixel / 6.0;

out.putPixel(x, y, pixel);

}

}

return out;

}

static public ImageAccess detectEdgeHorizontal\_Separable(ImageAccess input) {

int nx = input.getWidth();

int ny = input.getHeight();

ImageAccess out = new ImageAccess(nx, ny);

double rowin[] = new double[nx];

double rowout[] = new double[nx];

for (int y = 0; y < ny; y++) {

input.getRow(y, rowin);

doAverage3(rowin, rowout);

out.putRow(y, rowout);

}

double colin[] = new double[ny];

double colout[] = new double[ny];

for (int x = 0; x < nx; x++) {

out.getColumn(x, colin);

doDifference3(colin, colout);

out.putColumn(x, colout);

}

return out;

}

**Questão 2.1:**

**static public ImageAccess doMovingAverage5\_NonSeparable(ImageAccess input) {**

**int nx = input.getWidth();**

**int ny = input.getHeight();**

**double arr[][] = new double[5][5];**

**double pixel;**

**ImageAccess out = new ImageAccess(nx, ny);**

**for (int x = 0; x < nx; x++) {**

**for (int y = 0; y < ny; y++) {**

**input.getNeighborhood(x, y, arr);**

**pixel = 0.0;**

**for (int i = 0; i < 5; i++) {**

**for (int j = 0; j < 5; j++) {**

**pixel += arr[i][j];**

**}**

**}**

**pixel = pixel / 25.0;**

**out.putPixel(x, y, pixel);**

**}**

**}**

**return out;**

**}**

**Questão 2.2:**

**static public ImageAccess doMovingAverage5\_Separable(ImageAccess input) {**

**int nx = input.getWidth();**

**int ny = input.getHeight();**

**ImageAccess out = new ImageAccess(nx, ny);**

**double rowin[] = new double[nx];**

**double rowout[] = new double[nx];**

**for (int y = 0; y < ny; y++) {**

**input.getRow(y, rowin);**

**doAverage5(rowin, rowout);**

**out.putRow(y, rowout);**

**}**

**double colin[] = new double[ny];**

**double colout[] = new double[ny];**

**for (int x = 0; x < nx; x++) {**

**out.getColumn(x, colin);**

**doAverage5(colin, colout);**

**out.putColumn(x, colout);**

**}**

**return out;**

**}**

**static private void doAverage5(double vin[], double vout[]) {**

**int n = vin.length;**

**vout[0] = (vin[0] + 2.0 \* vin[1] + 2.0 \* vin[2]) / 5.0;**

**vout[1] = (vin[0] + vin[1] + vin[2] + vin[3]) / 5.0;**

**for (int k = 2; k < n-2; k++) {**

**vout[k] = (vin[k-2] + vin[k-1] + vin[k] + vin[k+1] + vin[k+2]) / 5.0;**

**}**

**vout[n-2] = (vin[n-4] + vin[n-3] + vin[n-2] + vin[n-1]) / 5.0;**

**vout[n-1] = (vin[n-3] + 2.0 \* vin[n-2] + vin[n-1]) / 5.0;**

**}**

**Questão 2.3:**

**static public ImageAccess doMovingAverage5\_Recursive(ImageAccess input) {**

**int nx = input.getWidth();**

**int ny = input.getHeight();**

**ImageAccess out = new ImageAccess(nx, ny);**

**double rowin[] = new double[nx];**

**double rowout[] = new double[nx];**

**for (int y = 0; y < ny; y++) {**

**input.getRow(y, rowin);**

**doAverage5\_Recursive(rowin, rowout, 0);**

**out.putRow(y, rowout);**

**}**

**double colin[] = new double[ny];**

**double colout[] = new double[ny];**

**for (int x = 0; x < nx; x++) {**

**out.getColumn(x, colin);**

**doAverage5\_Recursive(colin, colout, 0);**

**out.putColumn(x, colout);**

**}**

**return out;**

**}**

**static public void doAverage5\_Recursive(double vin[], double vout[], int k) {**

**int n = vin.length;**

**if (k == 0) {**

**vout[0] = (vin[0] + 2.0 \* vin[1] + 2.0 \* vin[2]) / 5.0;**

**vout[1] = (vin[0] + vin[1] + vin[2] + vin[3]) / 5.0;**

**doAverage5\_Recursive(vin, vout, k + 2);**

**} else if (k >= n-2) {**

**vout[n-2] = (vin[n-4] + vin[n-3] + vin[n-2] + vin[n-1]) / 5.0;**

**vout[n-1] = (vin[n-3] + 2.0 \* vin[n-2] + vin[n-1]) / 5.0;**

**} else {**

**vout[k] = (vin[k-2] + vin[k-1] + vin[k] + vin[k+1] + vin[k+2]) / 5.0;**

**doAverage5\_Recursive(vin, vout, k + 1);**

**}**

**}**

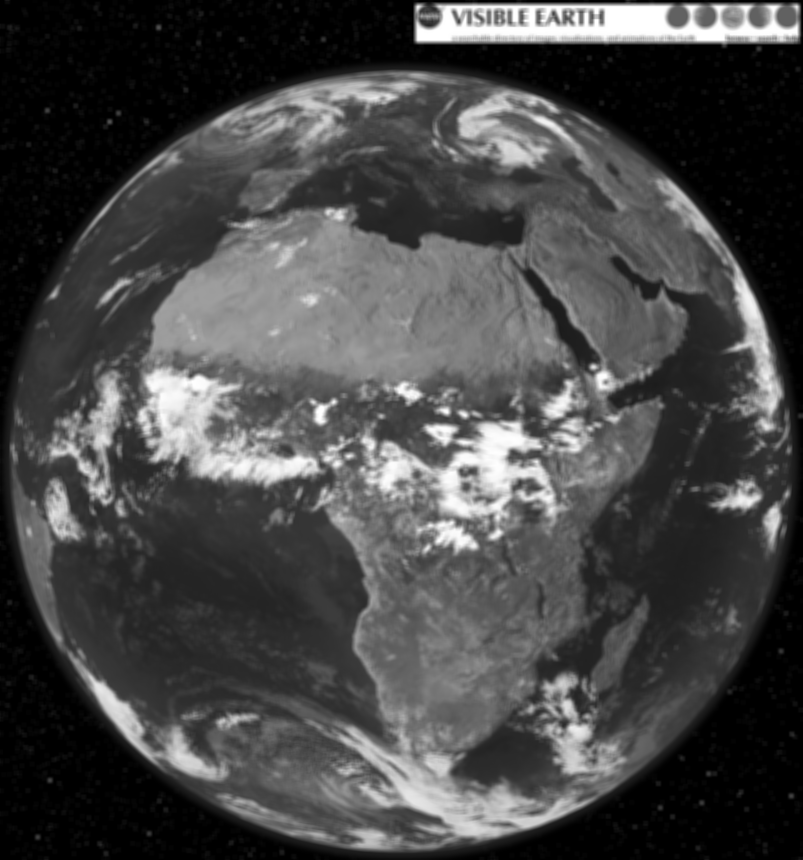
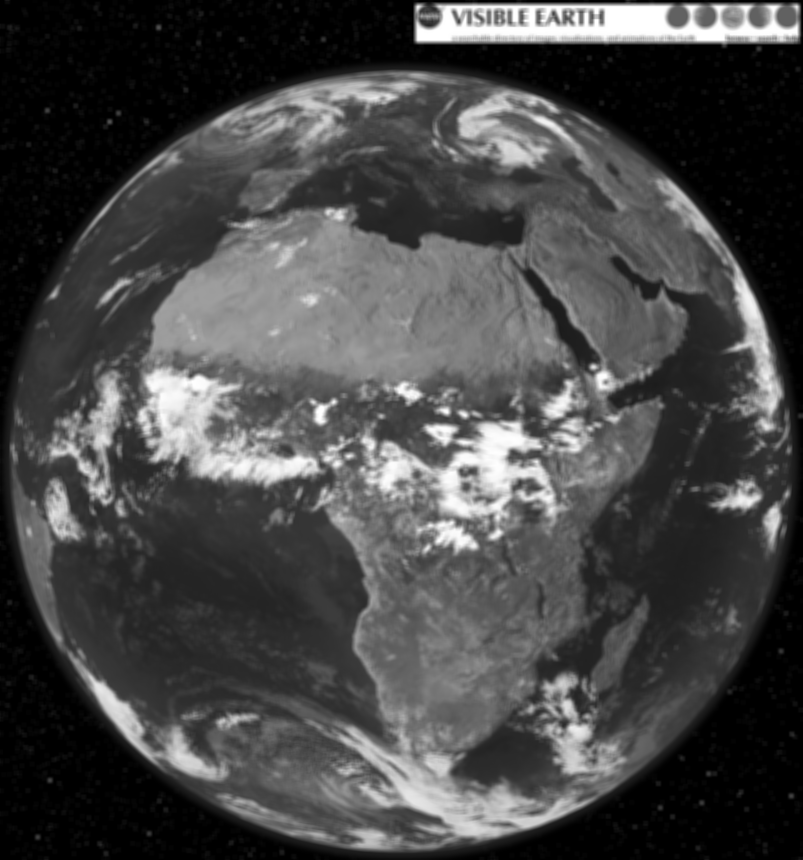
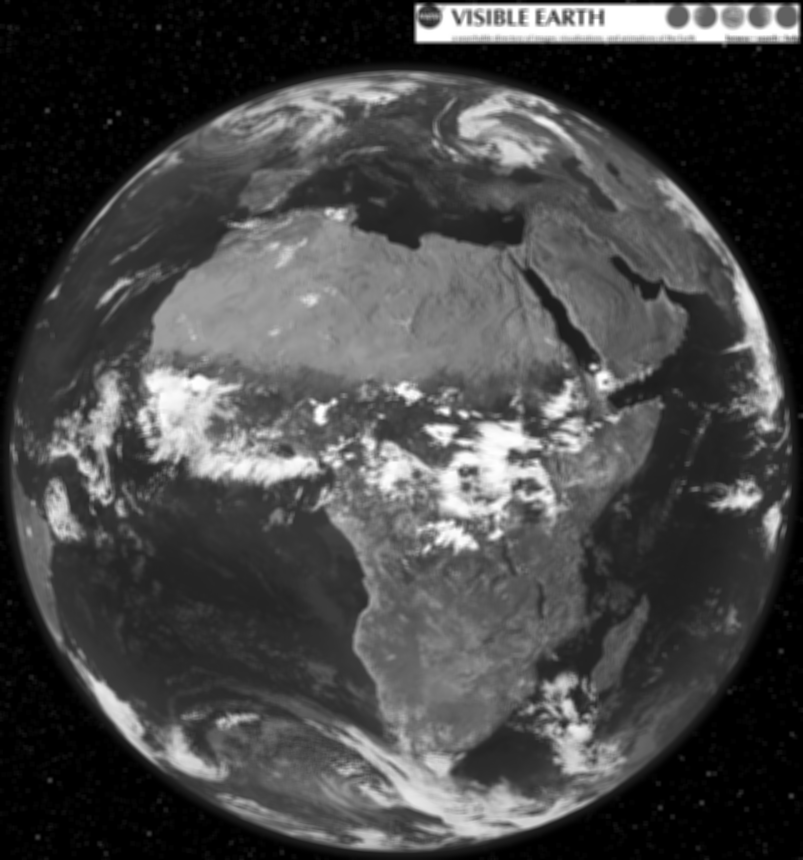
Média-móvel 5\*5 em africa.tif

|  | Tempo | Média | Mínimo | Maximo |
| --- | --- | --- | --- | --- |
| versão não-separável | 74ms | 71.44 | 0 | 255 |
| versão separável | 28ms | 71.44 | 0 | 255 |
| versão recursiva | 31ms | 71.44 | 0 | 255 |

Comparação:

As imagens são muito semelhantes. A versão Não Separável parece levemente mais borrada. A versão Não Separável é consideravelmente mais cara computacionalmente, a versão Separável é a mais barata e a Recursiva é marginalmente mais cara que esta.

(Non-sep/Sep/Rec)

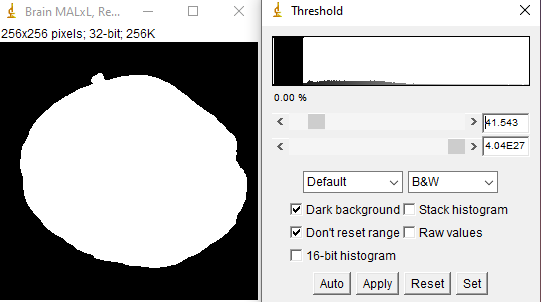


**Questão 3.1:**

Segmentando com o Operador Smoothing

L = 13

T = 41



Insira a parte do código escrito.

static public ImageAccess doMovingAverageL\_Recursive(ImageAccess input, int length) {

int nx = input.getWidth();

int ny = input.getHeight();

ImageAccess out = new ImageAccess(nx, ny);

double rowin[] = new double[nx];

double rowout[] = new double[nx];

for (int y = 0; y < ny; y++) {

input.getRow(y, rowin);

doAverageL\_Recursive(rowin, rowout, 0, length);

out.putRow(y, rowout);

}

double colin[] = new double[ny];

double colout[] = new double[ny];

for (int x = 0; x < nx; x++) {

out.getColumn(x, colin);

doAverageL\_Recursive(colin, colout, 0, length);

out.putColumn(x, colout);

}

return out;

}

static public void doAverageL\_Recursive(double vin[], double vout[], int k, int length) {

int n = vin.length;

if (k >= n) {

return;

}

if (k == 0) {

double sum = 0.0;

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

int index = Math.min(Math.max(i, 0), n - 1);

sum += vin[index];

}

vout[0] = sum / (double) length;

} else if (k >= n - length / 2) {

double sum = 0.0;

for (int i = n - length; i < n; i++) {

int index = Math.min(Math.max(i, 0), n - 1);

sum += vin[index];

}

vout[k] = sum / (double) length;

} else {

double sum = 0.0;

for (int i = k - length / 2; i <= k + length / 2; i++) {

int index = Math.min(Math.max(i, 0), n - 1);

sum += vin[index];

}

vout[k] = sum / (double) length;

}

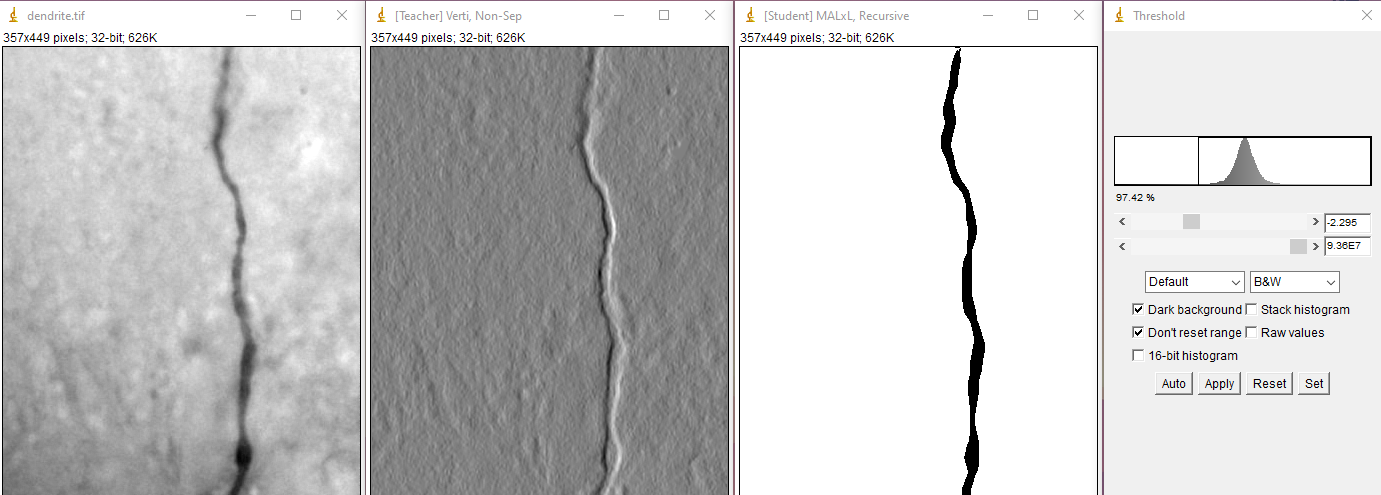
doAverageL\_Recursive(vin, vout, k + 1, length);

}

**Questão 3.2:**

Descreva o procedimento

1. Vertical Edge Non-Separable
2. Moving Average 13x13
3. Threshold

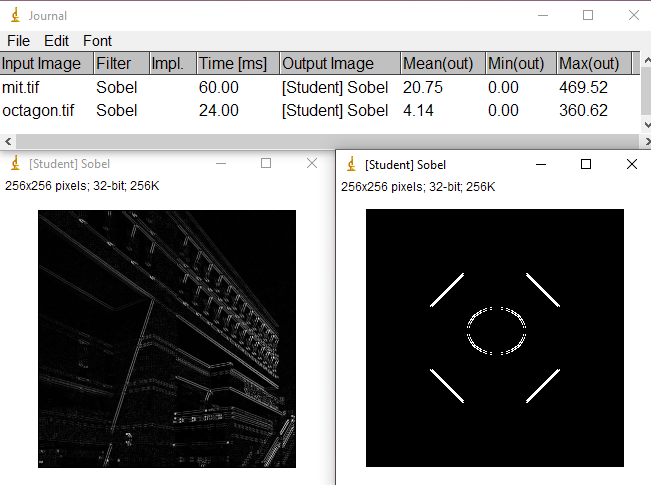


Insira a parte do código escrito

Código reutilizado: Questão 1.1 e Questão 3.1

**Questão 4:**

Operador Sobel



Insira a parte do código escrito para este exercício

static public ImageAccess doSobel(ImageAccess input) {

int nx = input.getWidth();

int ny = input.getHeight();

ImageAccess out = new ImageAccess(nx, ny);

double rowin[] = new double[nx];

double rowout[] = new double[nx];

double gx, gy;

// Apply horizontal Sobel kernel to rows

for (int y = 0; y < ny; y++) {

input.getRow(y, rowin);

doSobelHorizontal(rowin, rowout);

out.putRow(y, rowout);

}

// Apply vertical Sobel kernel to columns

double colin[] = new double[ny];

double colout[] = new double[ny];

for (int x = 0; x < nx; x++) {

out.getColumn(x, colin);

doSobelVertical(colin, colout);

out.putColumn(x, colout);

}

// Combine the results

for (int x = 0; x < nx; x++) {

for (int y = 0; y < ny; y++) {

gx = out.getPixel(x, y);

gy = out.getPixel(x, y);

double pixel = Math.sqrt(gx \* gx + gy \* gy);

out.putPixel(x, y, pixel);

}

}

return out;

}

static private void doSobelHorizontal(double vin[], double vout[]) {

int n = vin.length;

vout[0] = vin[0] - vin[2];

for (int k = 1; k < n - 1; k++) {

vout[k] = vin[k - 1] - vin[k + 1];

}

vout[n - 1] = vin[n - 2] - vin[n - 1];

}

static private void doSobelVertical(double vin[], double vout[]) {

int n = vin.length;

vout[0] = vin[0] - vin[2];

for (int k = 1; k < n - 1; k++) {

vout[k] = vin[k - 1] - vin[k + 1];

}

vout[n - 1] = vin[n - 2] - vin[n - 1];

}

**Questão 5:**

(Mesmo código da questão 3.1)

**static public ImageAccess doMovingAverageL\_Recursive(ImageAccess input, int length) {**

**int nx = input.getWidth();**

**int ny = input.getHeight();**

**ImageAccess out = new ImageAccess(nx, ny);**

**double rowin[] = new double[nx];**

**double rowout[] = new double[nx];**

**for (int y = 0; y < ny; y++) {**

**input.getRow(y, rowin);**

**doAverageL\_Recursive(rowin, rowout, 0, length);**

**out.putRow(y, rowout);**

**}**

**double colin[] = new double[ny];**

**double colout[] = new double[ny];**

**for (int x = 0; x < nx; x++) {**

**out.getColumn(x, colin);**

**doAverageL\_Recursive(colin, colout, 0, length);**

**out.putColumn(x, colout);**

**}**

**return out;**

**}**

**static public void doAverageL\_Recursive(double vin[], double vout[], int k, int length) {**

**int n = vin.length;**

**if (k >= n) {**

**return;**

**}**

**if (k == 0) {**

**double sum = 0.0;**

**for (int i = 0; i < length; i++) {**

**int index = Math.min(Math.max(i, 0), n - 1);**

**sum += vin[index];**

**}**

**vout[0] = sum / (double) length;**

**} else if (k >= n - length / 2) {**

**double sum = 0.0;**

**for (int i = n - length; i < n; i++) {**

**int index = Math.min(Math.max(i, 0), n - 1);**

**sum += vin[index];**

**}**

**vout[k] = sum / (double) length;**

**} else {**

**double sum = 0.0;**

**for (int i = k - length / 2; i <= k + length / 2; i++) {**

**int index = Math.min(Math.max(i, 0), n - 1);**

**sum += vin[index];**

**}**

**vout[k] = sum / (double) length;**

**}**

**doAverageL\_Recursive(vin, vout, k + 1, length);**

**}**